IEEE MICRO

APRIL 1992

Chips, Systems, Software, and Applications

Hot Chips III

Exploiting Parallelism for HIGHER PERFORMANCE

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE MICRO

Published by the IEEE Computer Society

April 1992

F E A T U R E S

Cover Image: Leo de Wys

Cover design: Design and Direction

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $23 in addition to IEEE Computer Society or any other IEEE society member dues; $39 for members of other technical organizations. This journal is also available in microfiche form.
Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices. Canadian GST#125634188.

* Submit six copies of all articles and special-issue proposals to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129 Torino, Italy; phone +39 11 556 4044; Compmail: d.delcorso; Internet: delcorso@polito.it; Bitnet: delcorso@itopoli
or
** Ashis Khan, Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086; Internet: ashis@mips.com.
Author guidelines for article submission appear in October 1992 *IEEE Micro*, pp. 28–29. Or, contact Managing Editor Marie English at address at left.

DEPARTMENTS

IEEE Computer Society
PO Box 3014, Los Alamitos, CA 90720-1264
(714) 821–8380

Expedited Delivery

is available to all members residing outside the USA, Canada, and Mexico. We invite you to take advantage of this service providing delivery of your magazine weeks earlier.

For information on this service and its cost, contact:

Expedited Delivery
IEEE Computer Society
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA
90720-1264 USA
Phone: (714) 821-8380
Fax: (714) 821-4010

***IEEE Micro* will reach you faster.**

In the mailbag

(LK: liked; DLK: disliked; LTS: like to see)

April 1991

LK: Editorial [is] different and diffuse; DLK: digressive editorial; LTS: different and diffuse magazine. (K.T., Bushehr, Iran)

June 1991

LK: Datawave multiprocessor—probably many other applications would benefit having multiple CPUs mesh-connected on a single die; LTS: more on event-driven interconnection schemes. (J.C., Warsaw, Poland)

August 1991

LK: Strong content; timely, useful. LTS: ...SCSI-ESDI, mass storage. (C.R.A., Searingtown, NY)

LK: The article on parallel processing; [This is a hot theme; you will find others in future issues.—D.D.C.] LTS: graphic stations, image processors. [They are coming.— D.C.C.] (T.M., Tama-City, Japan)

LTS: Microprocessor-based systems (controller, data acquisition ...) (A., Makkah, Saudi Arabia)

LK: Guest Editor's Introduction by Ken Sakamura: USA vs. Japan—differences in design style. Congratulations to 68040 authors! (J.R.C., Warsaw)

October 1991

DLK: The artificial intelligence review; LTS: Mips R4000. [It is in this issue.—D.D.C.] (J.B., Bedford, MA)

LK: Micro Review, "Computers and creativity." (Y.S., Hino-City, Japan)

LK: Micro Law (W.S., Zurich, Switzerland)

LK: The magazine as a whole; LTS: more articles about bioengineering. (O.S., Rio de Janeiro, Brazil)

LK: Well-balanced technical information. (J.M.S., Florianopolis, Brazil)

LK: Micro Law [and the] Li and Chen [article]; DLK: Intel 860 article, already outdated, too much [like a] "technical handbook." [An article on the next-generation i860 is coming, but I do not think the current 860 is outdated!—D.D.C.] (T.S., Tromso, Norway)

LK: The idea of register windows; LTS: the specification of P1154 standard. (R.K., Osaka, Japan)

Call for Papers

***IEEE Micro* seeks general interest manuscripts for 1992 and 1993 issues.**

Suggested topics include biological computing, VHDL design and workstations, multiprocessing, optical computing, microcomputing to aid the handicapped, and ICs for HDTV. Submit six copies of manuscripts to Editor-in-Chief Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129 Torino, Italy (Internet: delcorso@polito.it), **or** Associate Editor-in-Chief Ashis Khan, Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086 (Internet: ashis@mips.com).

Micro World

Hermes, the European space shuttle

Hubert Kirrmann

Asea Brown Boveri Research Center

CRBC. 1 CH-5405

Baden, Switzerland

Manned space travel has long fueled human imagination, science fiction books, and heated debates about its utility and funding.

Visionary authors like Jules Verne were well aware of the latter problem. In his book, *From the Earth to the Moon* (1865), Verne envisioned the difficulty of multinational funding for a moon projectile. The German confederation was short of money and could only contribute 34,285 florins. For the Vatican, the idea came too soon, just after the rehabilitation of Copernicus in 1822. Switzerland only granted 257 Swiss francs, since the Swiss did not feel they could tie in any trade relationship by shooting a cannonball to the moon. The English did not participate at all, since they considered the project incompatible with their principle of nonintervention. France was a driving force behind the Gun Club project, but the largest part of the funding came from Russia and the United States.

The European program

The funding of the European manned space program in 1992 follows this 1865 scheme. The Gun Club's successor is the ESA, or European Space Agency. On the table of negotiation are three major projects involving manned space flights: the European *Hermes* space shuttle , the *Columbus* manned station, and the *Ariane 5* rocket. The ESA counts 13 members and two associates, but the main actors are the economically stronger countries of France, Germany, and Italy. England is absent. Switzerland accounts for 2 percent of the funding. The unit of negotiation is not the dollar, but the ECU (European counting unit). One ECU equals approximately US$1.4.

The key component of the project is the *Hermes* shuttle,[1] which should carry a crew of three and a payload of 3 tons to low orbit. Its development cost is estimated at 6,200 million ECUs. *Hermes* is primarily designed to serve the *Columbus* Man-Tended Free Flyer (MTFF) space station scheduled for the year 2001. It could also serve the US *Freedom* space station. The *Columbus* attached, pressurized module (scheduled for 1998), which is the European contribution to the *Freedom* international program, will dock to the *Freedom* space station. Docking to the Russian *MIR* space station is also foreseen. Estimates of *Columbus*'s cost are 3,700 million ECUs. (*Freedom*'s budget is 14,000 million ECUs, or US$19.700 million.)

The *Columbus* precursor program foresees different *Spacelab* flights with the US space shuttle. This program involves auxiliary projects like the *Poem-1* polar orbit station or data relay satellites.

In a clever move, the ESA decided not to develop a special launcher for the *Hermes* shuttle, but to share the *Ariane 5* launcher with commercial satellites. Besides reducing development costs, this sharing produces two nice side effects. First, one can build on the experience accumulated with commercial satellites; second, the usefulness of the launcher does not come into question.

In fact, the unmanned European space program remains unchallenged. The Arianespace organization is responsible for the *Ariane* flights. *Ariane*'s clients include 90 percent of the world's satellite operators. They earned 20 million ECUs last year and received orders for 5,000 million ECUs. *Ariane*'s 100th stage just came off the assembly line. (It is the 50th rocket.) So, *Hermes* will be able to ride on that wave, although the launch capability of *Ariane 5* had to be boosted to put the 24.4 tons of *Hermes* in low orbit. The

Ariane 5 budget also increased to 3,500 million ECUs, but it seems secure now.

The ESA prepared a long-term plan, but the member states want to approve the yearly budget. Budget approval presents the main difficulty for the ESA—a similar situation to that of the US space station.[2] In November 1991 the ESA asked the member states to spend the impressive sum of 39,000 million ECUs and to make a yes or no decision on the *Hermes* program. To decrease the yearly budget, the ESA proposed to build only one shuttle, with the maiden flight postponed to the year 2002, one flight per year (instead of two), and operational capability by 2004. At their November 1991 meeting in Munich, the ministers preferred to adjourn the decision instead and await further studies.

The hesitation of the ministers just reflects the taxpayer's mood in the different countries. While 60 percent of the French favor manned space flights without reserve and consider it a matter of national pride, they would like the others to pay for it also. The Germans are more concerned with down-to-earth themes like environment and reunification costs, and the Italians simply lack the money.

But the usefulness of manned space flights is being questioned again—and not without reason. Indeed, the last 30 years since Yuri Gagarin's flight have shown that humans can contribute little in space. The Soviet *Progress* spacecraft demonstrated that automatic rendezvous in orbit was reliable, and the *Luna* probe brought moon rocks back to the earth at a fraction of the *Apollo* project's costs. Human flights are restricted to low earth orbit, but money is made on the geostationary orbit. And for astronomical or earth observation, nothing is as disturbing as a human being moving or sneezing inside the spacecraft.

But for the public, manned space travel is tied to emotion, pride, and dreams, and it is ready to pay for them. Despite the end of the Cold War, many see space exploration as a competition, not unlike the America's Cup race. And this makes manned space travel probably the most important psychological motor to the development of spacecraft.

In 1992, six Europeans plan to enter space aboard foreign spaceships: three US shuttle flights (*Discovery* once and *Atlantis* twice) with *Spacelab* on board and two Russian missions to the *Mir* station. The real question is: Does Europe need its own shuttle when space tickets are already on sale?

The usefulness of manned space flights is being questioned again—and not without reason.

And the Russians argue that the European shuttle already exists: They would like to sell their *Buran* (Blizzard) shuttle as an alternative to *Hermes*. This shuttle already performed its automatic maiden flight. *Buran*'s next flight is scheduled for 1993, but budget restrictions and the political situation could delay it.

It is unlikely that the ESA will accept the Russian offer, because *Hermes* benefits are not found in operating it but in building it. The project allows the European industry to become acquainted with new space technologies. It would give work to 16,000 persons in the next 10 years.[2] It should form links between countries in another European project and possibly extend them to the Eastern countries. The Russians have already offered precursor flights on their simulators and training facilities. For industry and universities, *Hermes* presents a challenge, a test bank, and an attraction pole for qualified engineers.

Is a space shuttle the correct answer to economic space flights? The fact that Russia also built a shuttle is no proof of its usefulness: In strategic games, one covers each move of the adversary. The US shuttle failed in at least two aspects. It was neither cheaper nor easier to operate than expendable rockets. And because the US shuttle must be manned, one failure delayed the whole US space program for two years. *Hermes* and *Buran* flights do not need to be manned, but a failure, even in the first unmanned mission, could stop the program as well.

The next generation is already on the drawing board: one- or two-stage spaceships that use air during atmospheric ascent and switch to rocket mode to reach orbit, like the British/Russian *Hotol*, the German *Saenger*, or the French *Star-H*. But the way to such spacecraft comes from mastering the technology, and this shall be worth the cost of *Hermes*. The fact that the ESA budgeted only one *Hermes* spaceplane shows that it is nothing more than a prototype.

Fault-tolerant multiprocessor

For the computer architects, the most challenging part of the *Hermes* project is its on-board computer, called the SEF.[3] This fault-tolerant system, developed by Matra Marconi Space (Toulouse, France), will be the core of the *Hermes* avionics and support guidance, communication, and the mission itself.

The fault-tolerant computer system consists of a pool of four computers interconnected by serial links. It looks very similar to other avionics computers like the US space shuttle Primary Computer (1974), the SIFT (1978), or the FTMP (1978) computers. The architecture has not changed much in the last 17 years. Why should it? *La fonction fait l'objet.*

The most critical function supported by the SEF is the guidance, navigation, and control (GNC) of the spaceplane. GNC is normally a repetitive task: Read the input sensors, process the input data, and generate the command to the actuators. The GNC computer uses a high-performance RISC processor (the Sun Microsystems Sparc is a candidate) with 2 Mbytes of memory to provide 4 MIPS of processing power. The boards of the GNC computer interconnect via a Nubus (IEEE Std. 1196).

The main input sensors are the inertial navigation system, the global positioning system, and the radio altimeter. A set of sensors connects to each of the four GNC computers through a dedicated bus. The critical communication link to the *Ariane 5* system is duplicated.

To cover the time-critical flight phases, the computer masks errors rather than using time-consuming recovery methods. To this effect, four processors execute the GNC algorithms in parallel. The processors are synchronized to operate on the same input data set to ensure that they do not diverge. Each computer reads its inputs and sends their values to the other three computers over the 7-Mbps serial interprocessor link. The computers reach a consensus on the input data, process the data, and broadcast the result, so as to reach a consensus on the output value. Only then is the value forwarded to the actuators.

To offload the application processors from synchronization and matching, a dedicated processor, called Data Manager, handles the four serial interprocessor links between the computers. These links, called the Interprocessor Network, provide a reliable clock synchronization with a 20-ms period. This new approach responded to recognition that synchronization is a critical and time-consuming function, especially when executing Byzantine agreements.

This arrangement can still fail because of common-mode errors. The most obvious is that the same software error may affect all computers. Therefore, the programs running in the different processors may be diversified, for example, written by different persons. So it becomes unlikely that the same error will affect all computers. This technique is called *N*-version programming. It is used today, for instance, in the *Airbus 320* computers.

This is also a new approach for another reason. Previous projects, such as the US space shuttle, did not foresee software diversity or let it execute on a distinct computing system. A representative prototype based on functional models of this fault-tolerant computer pool architecture has already been developed by Matra Marconi to validate the concept. Final space qualification will be the real challenge of the SEF development, especially with respect to the tools and methods involved. Too often, fault-tolerant computers have been a pill in search of a disease. The SEF offers a unique opportunity to apply the theory of fault-tolerant computing where it is really needed. This is one of the merits of the *Hermes* project.

What is the future of *Hermes*? The Greeks named Hermes the god of eloquence, trade, and thieves. The future will show which name really applies, if not all three.

Acknowledgment
My thanks go to A. Blanc for his collaboration.

References
1. "Europeans Facing Major Hurdle in Implementing Long-Term Space Plan," *Aviation Week*, June 17, 1991, pp. 96-99.
2. J. Feustel-Buechl, "Wings for European Spaceflight—the Future of Space Transportation," European Space Agency, Paris, 1991, pp. 21-28.
3. A. Blanc and A. Mosnier, *Proc. AIAA/NASA Second Int'l Symp. Space Data Systems,* AIAA-90-5028, 1990.

Reader Interest Survey
Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 186	Medium 187	High 188

Micro Review

Consciousness

Richard Mateosian

2919 Forest Avenue

Berkeley, CA

94705-1310

(510) 540-7745

Regular readers of this column have seen several reviews of books about human mental processes. In October 1988 I talked about Johnson-Laird's frustrating *The Computer and the Mind.* Last August I reviewed Penrose's *The Emperor's New Mind,* a work whose main conclusion about the inability of artificial intelligence (AI) to replicate human behavior depends on an as yet undiscovered theory of quantum gravitation. Then last October I looked at Boden's *The Creative Mind,* a work that takes a contrasting view of AI and fails to disagree completely with Penrose only because of a last-minute appeal to human chauvinism.

This time I have looked at a work that attempts no less a task than the complete explanation of consciousness. It is a serious and scholarly exploration of the great central problem of psychology and philosophy, the mind-body problem. I predict that many people will buy it, but few will read it, and fewer still will understand and adopt the viewpoint that it teaches.

Consciousness Explained, Daniel C. Dennett (Little, Brown, Boston, 1991, 524 pp., $27.95)

The first thing I have to tell you about this book is that its explanation of consciousness, by the author's own admission, is sketchy. Dennett tries to explain the most important physiological and psychological facts and to answer the most widely known philosophical arguments that contradict his point of view. At many points, however, he simply gives the shape of the theory, leaving to further research the fleshing out of details. As he says in his appendix for scientists, in which he proposes several experiments,

> Since as a philosopher I've tried to keep my model as general and noncomittal as possible, if I've done my job right, these experiments should help settle only *how strong* a version of my model is confirmed; if the model were entirely disconfirmed, I would be well and truly refuted and embarrassed.

Dennett is a wonderful storyteller. His ability to make points and cut through jargon with well-chosen analogies, anecdotes, and caricatures is breathtaking. Again and again as the going gets rough, he finds a way to bring the discourse back to an arena in which the reader feels at home. For example, in discussing color vision, a favorite topic in philosophical discourses on human mental processes, he mentions the Rosenbergs' torn Jell-O boxes. Two spies could identify themselves to each other by producing the torn halves of a Jell-O box. Neither half's pattern has any intrinsic significance, but each matches the other perfectly. This story cuts through philosophical jargon that goes back hundreds of years and makes clear immediately Dennett's view of why there are colors.

In *The Selfish Gene* (Oxford, 1976), Richard Dawkins coined the term *meme* to describe complex idea units (like wheel, alphabet, calculus, the *Odyssey,* the theme from the slow movement of Beethoven's Seventh Symphony). Memes are central to Dennett's view of consciousness. He says,

> Human consciousness is *itself* a huge complex of memes (or more exactly, meme-effects in brains) that can best be understood as the operation of a "von Neumannesque" virtual machine implemented in the parallel architecture of a brain that was not designed for any such

0272-1732/92/0400-0006$03.00 © 1992 IEEE

activities. The powers of this virtual machine vastly enhance the underlying powers of the organic hardware on which it runs. But at the same time many of its most curious features, and especially its limitations, can be explained as the by-products of the *kludges* that make possible this curious but effective reuse of an existing organ for novel purposes.

This certainly sounds like strong AI, the doctrine so vehemently opposed by Penrose. If you allow this definition to stand, you have to accept conscious machines or, alternatively, Dennett's assertion that we're all zombies. That is, Dennett believes there is no consciousness of the mysterious (epiphenomenal) sort posited by people who say that human beings must be more than "mere" Turing machines, no matter how closely machines can simulate their behavior. These are views, says Dennett, that do not deserve to be discussed with a straight face.

One of the most important things to learn from Dennett's book is how to apply his highly counterintuitive point of view. Most people who introspect about their minds tend to picture a control room in which a self gathers together sensory inputs and remembered information. The self uses these to make decisions and issue commands to the mechanisms that control speech, movement, and so forth. For example, in speaking of the effect of the "blind spot" where the optic nerve passes through the retina, many writers say that the brain fills in the missing part of the field of view. This view makes no sense unless there is an internal projection of the visual field and an internal observer viewing that projection in the control room. Descartes placed the control room in the pineal gland, but no serious modern thinker believes the control room model corresponds to actual brain function.

I don't want to give a detailed explanation of Dennett's replacement for this model. To me, his view of a person is like a roomful of monkeys at typewriters putting out a single newsletter. He regards the self as a construct, like center of gravity, whose usefulness breaks down if you get too close. In fact, he uses the term "narrative center of gravity."

While most thinkers reject the Cartesian control room model, many let it enter implicitly into their arguments, especially in "thought experiments." Thought experiments are parables, like Searle's Chinese Room (see Micro Review, August 1991). The philosopher devising the thought experiment asks you to imagine a situation that is possible in principle but usually impossible in practice. Then you are asked to follow a handwaving argument that leads to the point the philosopher is trying to make. Dennett ridicules a few notorious thought experiments. These exercises are good examples of the application of his model.

This review of Dennett's densely packed 500 pages is necessarily sketchy and incomplete, and it may not give much inkling of the excitement I felt while reading it. If you are interested in this subject, you should invest the time necessary to read and understand Dennett's book.

Macintosh utilities

Every year after the MacWorld Expo in San Francisco in January, I receive a large number of Macintosh programs to review. This year there seemed to be a better selection of utility programs than I've seen in previous years.

Now Utilities, Version 3.0 (Now Software, 520 S.W. Harrison St., Suite 435, Portland, OR 97201; (503) 274-2800, $129)

The Now Utilities is a package of 10 programs. The company has tried to cover all the bases, but other manufacturers provide better products for some of the functions. I think the best parts of the package are the Now Menus and Super Boomerang.

Now Menus allows cascading up to five levels. This is ideal for use with the Apple menu under System 7, since the most natural way to organize Apple menu items is in nested folders. Super Boomerang remembers applications and documents that you have used recently and makes them instantly available by slightly modifying the operation of the file-selection dialogs used by all Macintosh application programs.

WYSIWYG Menus is another useful program. It causes each entry in a font-selection menu to appear in characters from the font named by the entry. Of course, this program has a few drawbacks. For example, the names of fonts like Symbol or Zapf Dingbats are rendered in greek or in meaningless pictures.

Startup Manager lets you determine which startup programs to use and in which order. This program can be very helpful in debugging startup conflicts.

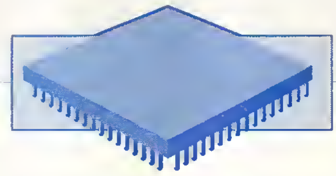
The other programs of the Now Utilities are also useful, but you don't have to use them. Each of the utilities can be installed separately. Even if you only use a few of them, you'll still get your money's worth.

Alsoft Power Utilities (Alsoft, Inc., PO Box 927, Spring, TX 77383-0927; (713) 353-4090, $129)

Alsoft's package of seven utilities is more narrowly focused than Now's. Four of the utilities pertain to disk operation. The others are a menu utility that is similar to Now Menus, a screen dimmer, and the partially obsolete (for System 7 users) Master Juggler.

Disk Express II keeps the files of your hard disk stored as efficiently as possible. It reorganizes files on demand or once per day in the background. It also removes fragmentation and keeps frequently used files together. It performs its job one file at a time, so that it is interruptible and little damage is done if it crashes.

continued on p. 79



Guest Editor's Introduction

Hot Chips III

Norman P. Jouppi

Digital Equipment Corporation

The annual Hot Chips Symposium presents the most current and exciting chip developments, as well as work in progress. The recent third meeting again boasted record attendance and required moving to the largest auditorium at Stanford University. The authors of seven of the most interesting and technically solid presentations were invited to submit papers for this special issue of *IEEE Micro*. Six authors agreed to submit papers, three were able to, and two appear here. In addition, this issue also carries an article detailing a recently developed and indisputably "hot" chip that was not presented at the symposium.

A theme running through the three special issue articles is that of exploiting parallelism for higher performance. Each product exploits parallelism in a different way.

Authors of the first article explain how the Mips R4000 exploits instruction-level parallelism through superpipelining. Superpipelining refers to the further pipelining of what are normally fundamental single-cycle operations in a pipelined machine. For example, on-chip cache access usually occurs in one cycle in pipelined microprocessors (not counting the usual cycle for address calculation). In contrast, the R4000 cache access is pipelined into three stages: two for cache access and one for tag comparison and control. The R4000 also provides support for a moderate degree of multiprocessing.

Next is a message-driven processor used in the J-machine at MIT. The message-driven processor exploits parallelism through large-scale and fine-grain multiprocessing. Each processor can deliver a message and dispatch a task to handle it with a latency of under two microseconds. In comparison, this is within a order of magnitude of the time required for a main memory access in most computers. The architecture and hardware design of the J-machine supports up to 4,096 processors!

The third article describes the 88110 from Motorola. This microprocessor makes use of instruction-level parallelism by issuing multiple independent instructions in the same cycle (that is, a superscalar approach). The 88110 adds graphics support and a floating-point register file to the 88000 architecture. Many organizational features add to performance, such as the out-of-order issue of stores, nonblocking caches, and 10 independent functional units. All but one of the functional units can begin a new operation each cycle. The 88110 also provides support for a moderate degree of multiprocessing.

Hot Chips IV is already in the planning stages. It takes place August 9-11, 1992, at Stanford University. If you'd like to submit a presentation, contact Bob Miller at (510) 642-6037 (bmiller@ginger.berkeley.edu). If you'd like more information about the 1992 symposium, contact Glen Langdon at (408) 459-2212 (langdon@cse.ucsc.edu).

I thank those reviewers who helped referee submissions with an amazing one- or two-week turnaround, and all the other people who helped with this issue.

0272-1732/92/0400-0008$03.00 © 1992 IEEE

Norman P. Jouppi is a member of the research staff at Digital Equipment Corporation's Western Research Lab in Palo Alto, California, and a consulting assistant professor in the Electrical Engineering Department of Stanford University. Previously, he was one of the principal architects and implementers of the Stanford Mips processor. While at Western Research Lab he was the principal architect and implementer of the Multi Titan CPU. His current research interests include computer architecture, VLSI design, and VLSI CAD tools.

Jouppi received BSEE and MSEE degrees from Northwestern University, Evanston, Illinois, and the PhD degree in electrical engineering from Stanford University. He was a National Science Foundation Fellow from 1980 to 1983 and is a member of the IEEE, Computer Society, and ACM.

Readers with questions concerning this special issue may contact Jouppi at Digital Equipment Corporation, 250 University Ave., Palo Alto, CA 94301; jouppi@pa.dec.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

High 150 Medium 151 High 152

Call for Articles

Advanced Packaging and Interconnect Technology

April 1993 Special Issue

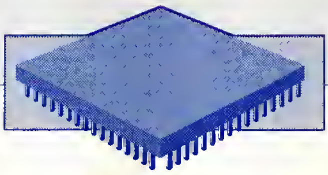
IEEE Micro

The April 1993 special issue of *IEEE Micro* will feature articles on advanced packaging and interconnect technology, as a companion issue to the April 1993 *Computer* special issue on multichip modules. The guest editor solicits manuscripts in the areas of

- critical packaging trends and issues;
- substrate and package technologies—for example, flexible, glass, or diamond substrates, few-chip packaging, or 3D packaging;
- attachment, bonding, and connection technologies, including fine-pitch surface mount, laser applications, known-good die, and interconnection trade-off analysis;
- system-level issues such as test, performance modeling, cooling, and EMI; and
- materials technology.

Authors should submit six copies of an original manuscript **by July 1, 1992;** notification of decisions is set for October 1, 1992; and the deadline for submission of the final version of each manuscript is December 1, 1992. For author guidelines, contact Clair Azada, Computer Society West Coast Office, (714) 821-8380.

Direct submissions and questions to Guest Editor David Misunas, MCC, 12100 Technology Blvd., Austin, TX 78727, phone (512) 250-3045, fax (512) 250-3045.



The Mips R4000 Processor

Computer architects estimate that the current generation of 32-bit machines will be obsolete by 1997. The R4000 employs a 64-bit architecture, using 64-bit registers and generating 64-bit virtual addresses. Superpipelining techniques allow it to process more instructions simultaneously than the previous generation of microprocessors. Specmark ratings indicate it performs higher than other single-chip microprocessors.

Sunil Mirapuri

Michael Woodacre

Nader Vasseghi

Mips Computer Systems

The R4000 is a highly integrated, 64-bit RISC microprocessor that provides a simple solution to the increasing demands on the size of address space, while maintaining full compatibility with previous Mips processors. Its primary features include

- on-chip CPU, FPU, MMU, primary caches, and system interface logic (See Figure 1),[1]
- superpipelining techniques,
- on-chip secondary cache control logic with a flexible interface,
- a programmable system interface for high-performance multiprocessor servers and low-cost desktop systems,
- flexible multiprocessor support, and
- 1.2 million transistors implemented in CMOS technology.

In addition, the R4000's single-chip implementation makes it easier to scale the clock as technology improves. According to SPEC benchmark tests, it achieves the highest performance of any microprocessor chip.

A 64-bit architecture

With programs growing by one-half to one bit of address space per year,[2] a greater than 32-bit address space should be useful by 1993 and required by 1997. In creating the 64-bit R4000, designers extended the R3000 architecture by increasing the data word size and virtual address space. This design entailed widening the machine registers and data paths, and sign-extending 32-bit data when loading into registers. Since certain operations work differently on 64-bit data than on sign-extended 32-bit data, we added additional instructions for 64-bit data, including integer loads, stores, adds, subtracts, shifts, multiplies, divides, and coprocessor moves.

The chip also supports a 64-bit virtual address space with wide virtual address data paths. It stores 32-bit addresses as 64-bit entities in sign-extended form and stores the results of address computation on these entities in sign-extended form. Thus it continues to support the previous 32-bit architecture's addressing.[3]

The hardware cost of extending the architecture to 64 bits was about 7 percent of the die

0272-1732/92/0400-0010$03.00 © 1992 IEEE

area. A longer, 64-bit ALU stage represents the cycle time speed penalty.

CPU pipeline

The R4000's eight pipeline stages allow it to process more instructions at once than can the R3000's five-stage pipeline.[4] Superpipelining has split the instruction and data memory references across two stages. Consequently, we could distribute the logic more evenly across pipeline stages. (See Figure 2.) The single-cycle ALU stage takes slightly more time than each of the cache access stages.

Although the superpipeline increases the cycles per instruction due to longer branch and load delays, it greatly improves the achievable cycle time. Future increases in cache size will not require a fundamental redesign of the superpipeline. We considered superscalar design as another way to increase instruction-level parallelism, but our studies showed that with current technology the chip could perform higher with a less complex superpipeline.

Figure 3 on the next page shows optimal pipeline movement, completing one instruction every internal clock cycle. The internal, or pipeline, clock rate of the R4000 is twice the external input, or master, clock frequency.

The processor accesses the instruction cache during the instruction first (IF) and instruction second (IS) stages, with a new cache access starting every cycle. The MMU translates the instruction virtual address into a physical address during these stages. The instruction bits available at the beginning of the register file (RF) stage are decoded and used to access the register file. Also at this time, the tags read from the instruction cache are compared with the physical address to determine whether the instruction cache access was a hit. If so, the instruction can advance to its execution (EX) stage. For nonmemory operations, the instruction's result is available by the end of the EX stage.

In the data first (DF) and data second (DS) stages, the R4000 accesses

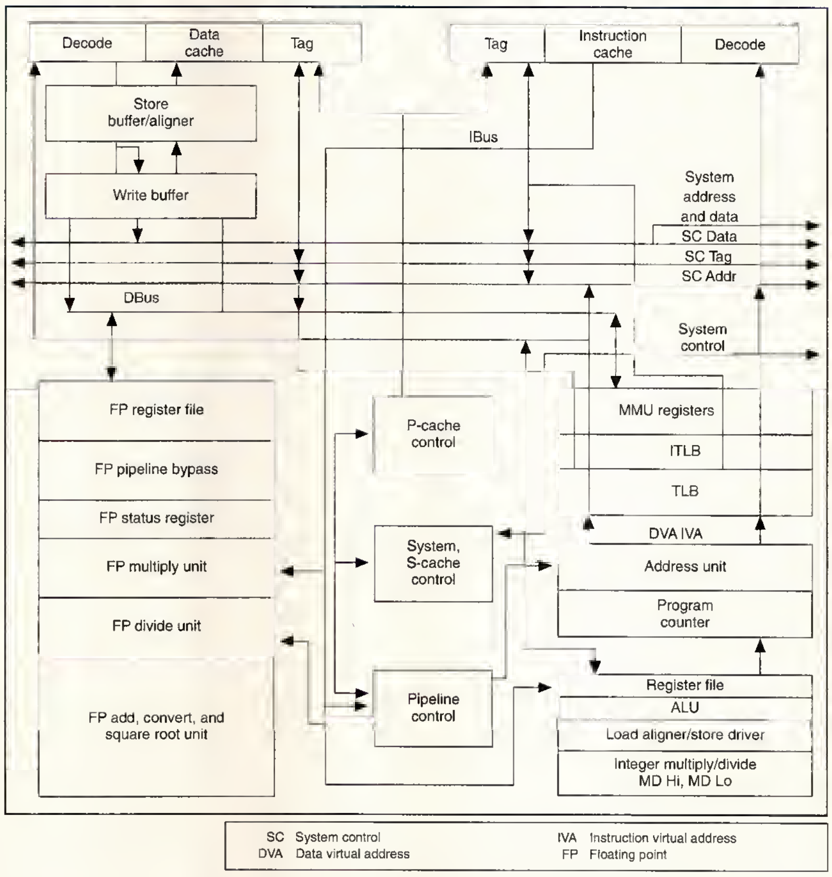


Figure 1. R4000 internal block diagram.

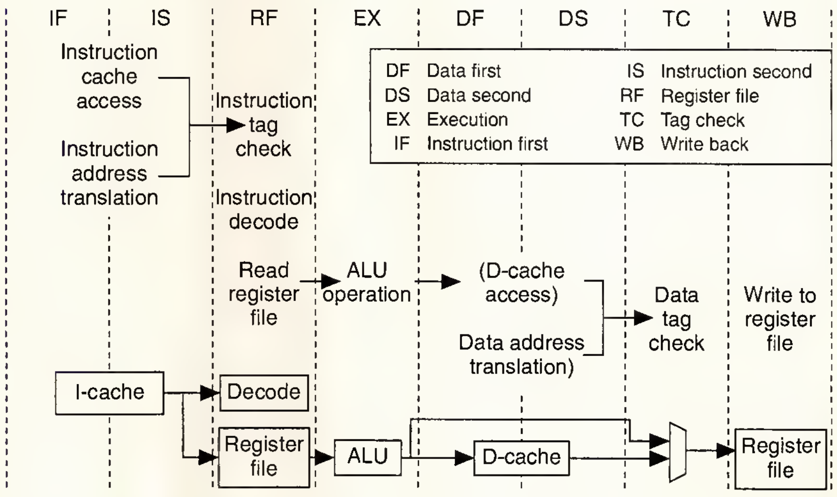


Figure 2. R4000 pipeline activities.

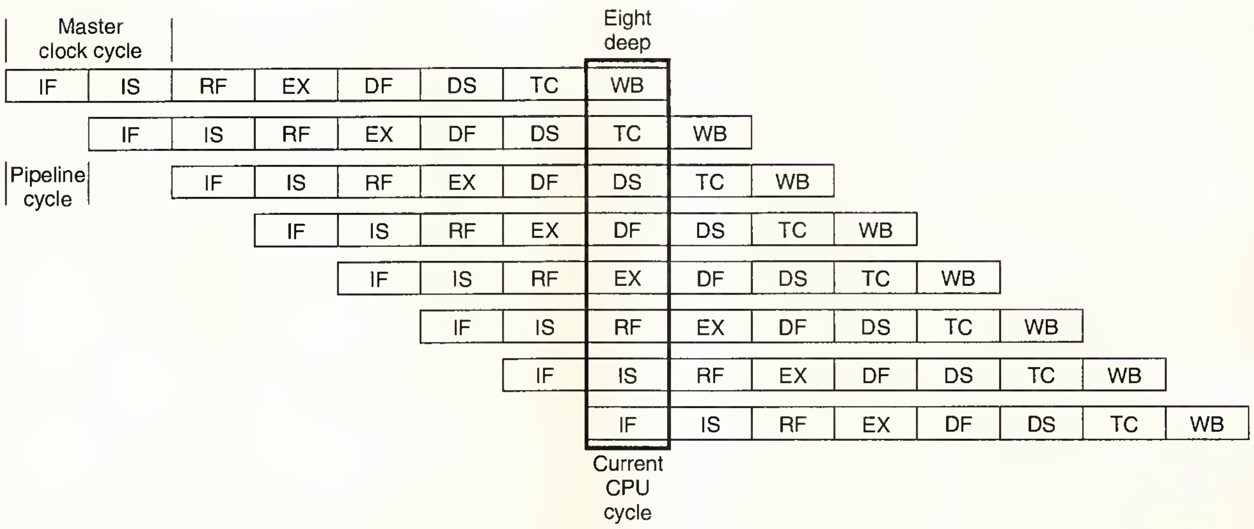


Figure 3. R4000 pipeline and instruction overlapping.

the data cache, with a new access starting every cycle. The MMU translates the data virtual address into a physical address during these stages. In the tag check (TC) stage, the R4000 compares the data tags from the cache tag array with the translated address to determine if the data cache access was a hit. For stores, if the tag check passes in TC, the data travel to the store buffer and enter the data cache the next time cache bandwidth is available. Instructions finally go to the write back (WB) stage where the data are written to the register file if necessary.

Load interlocks and branch instructions disrupt the normal flow of the pipeline. For loads, the data are not ready until the end of the cache access in the DS stage. If any of the two instructions after a load use the result of the load in their EX stages, the hardware interlocks and slips. As shown in Figure 4, during the slip the DF, DS, TC, WB stages of the pipeline advance while the IF, IS, RF, EX stages do not. For the load interlock, this permits the load instruction to advance and complete its cache access, while the instruction that depends on the load remains in the EX stage.

The result of a branch condition check and a branch target address calculation are not known until the end of the EX stage. (See Figure 5.) By that time, up to three subsequent instructions have entered the pipeline. If the branch is not taken, the processor can continue to execute all instructions that have entered the pipeline with no penalty. If the branch is taken, the processor accesses instructions at the branch target address. For taken branches, the Mips architecture allows one instruction after the branch to complete before executing the branch target instruction. The other two instructions that have already entered the pipeline are nullified. We considered a branch target scheme that prefetches instructions from both paths of a branch, producing a smaller branch penalty. However, implementation constraints required the simpler approach without a prefetching scheme.

Results of instructions that have completed their execution, but have not yet written their results into the register file, may be bypassed as operands for subsequent instructions.

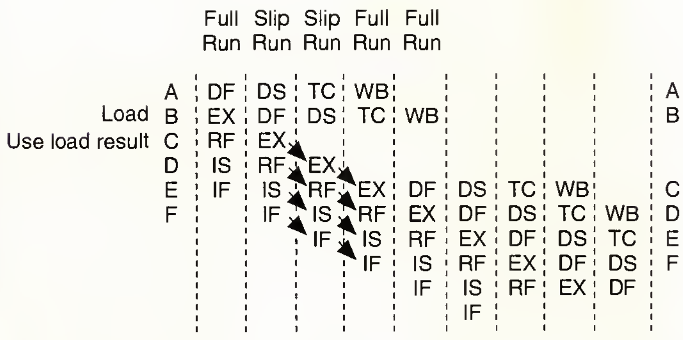


Figure 4. Load interlock/slip cycle.

Integer data path

The R4000's 64-bit execution unit includes a 64-bit register file, load aligner, ALU, shifter, multiplier, and divider. The 64-bit data path supports extended addressing without the use of long pointers or segment registers.[5]

The ALU stage, EX, was a speed-critical path. To shorten the cycle time, the ALU comprises an adder and a logical unit. The 64-bit, carry-select adder manipulates all 32-bit operands as sign-extended, 64-bit operands. It also performs address calculations for loads, stores, and branches, and is used in integer multiply and divide.

R4000 provides hardware support for integer multiply and divide. It uses a 2-bit Booth algorithm for integer multiplication and breaks each iteration into four stages: Booth decoding, multiplicand selection, partial product generation, and product accumulation. The carry-save adder (CSA) adds intermediate partial products, and two separate 64-bit registers Hi and Lo store the final product.

The multiplier cycles at twice the pipeline clock frequency to produce two sums for each pipeline cycle. Since the R4000 uses a CSA, the multiply results are in a sum-and-carry form and must be combined through full carry propagation. The integer ALU performs this operation when the result moves to the general registers. Integer multiply latency is 10 pipeline cycles for 32-bit operations and 20 pipeline cycles for 64-bit operations.

Divides use a 1-bit-per-iteration, nonrestoring algorithm. This algorithm leaves the quotient in a signed-digit form that must be converted back to a binary representation and possibly corrected at the end of the divide. Divides use the main integer adder for the remainder add or subtract operations, thus preventing the instructions from entering the pipeline during a divide. The implementation takes two pipeline cycles per iteration; each iteration resolves 1 bit of dividend. The latencies are 69 pipeline cycles for a 32-bit divide and 133 pipeline cycles for a 64-bit divide operation. We found this performance sufficient, due to the infrequent occurrence of the integer divide operations.

The integer shifter performs immediate or variable shifts from zero to 63 places. We designed the shifter to shift up to 32 bits in one cycle, making it half the size of a 64-bit shifter. To accomplish shifts greater than 32 bits, the pipeline slips for one cycle while forcing a 32-bit shift in the EX cycle. In the next cycle, the shifter performs the remainder of the shift. A trade-off between area and performance led to this decision.

The register file is a 32-entry by 64-bit array with two read ports and one write port. It can read and write in the same cycle. In the case of reading and writing the same location in the same cycle, the R4000 provides local bypassing of the write data into the read bus.

Floating-point unit

The FPU implements the IEEE Std 754-1985.[6] Its three functional units—multiplier, adder, and divider—operate on single- and double-precision operands. While the FPU executes a multicycle operation, the CPU pipeline can continue in parallel until the FPU detects a data or resource dependency. It can transfer data directly to or from the CPU or cache memory. The FPU executes up to three instructions concurrently, one per functional unit. It retires only one instruction per cycle.[7]

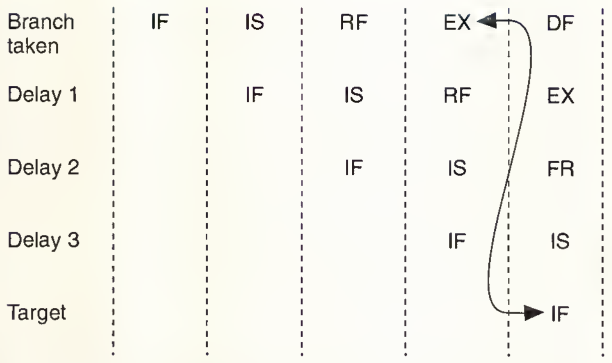


Figure 5. Branch delay.

The floating-point multiplier (see Figure 6 on next page) uses a modified Booth algorithm that scans four overlapping groups of 3 bits at once. Thus 8 bits of the multiplier operand can retire with each iteration. The mantissa portion of the multiply array uses four CSAs in a pipeline fashion. The multiplier pipeline includes four stages:

- Booth encoding and multiplicand selection,
- partial sum-and-carry generation of selected multiplicands,
- partial product summation of the previous stage result with the previous iteration result, and
- guard, round, and sticky-bit generation.

In the cycle following the last iteration of the multiply, the sum and carry from the multiplier array travel to the floating-point adder to produce the final rounded product.

The multiplier cycles at twice the pipeline clock frequency, so each iteration through the multiplier takes only half a pipeline cycle. R4000's high-speed operation demands that the multiplier array use a two-phase design approach. To reduce the clock skew in this region, the multiplier uses stronger clock drivers (with lower fanout). These drivers allow more aggressive latch designs with improved set-up times, and thus reduce overhead. All CSA and Booth multiplexers use dynamic logic design due to speed criticality.

The floating-point multiply latency is seven pipeline cycles for single-precision and eight for double-precision operations. The repeat rate is three pipeline cycles for single precision and four for double precision.

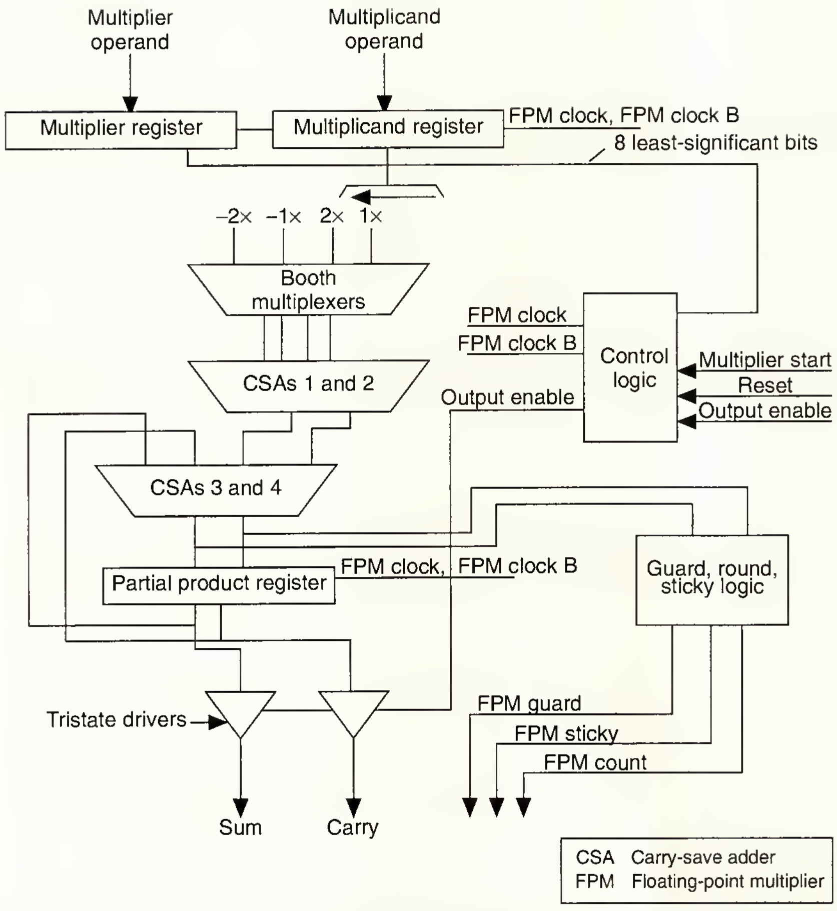


Figure 6. Block diagram of the floating-point multiplier.

The floating-point adder (Figure 7) processes one add or subtract in four pipeline cycles and starts a new operation every three pipeline cycles for both single- and double-precision operations. The adder also assists the multiplier and divider for cleanup operations, such as rounding, and final result computation.

To provide necessary bandwidth to support a two-staged, pipelined multiplier (as seen by the adder), we designed the adder to process a pair of double-precision, multiply-and-add instructions every four cycles.

The adder comprises four stages:

- unpack,
- mantissa add,
- result rounding, and
- mantissa shift (alignment/normalize).

The adder has two data entry paths. One accommodates the normal source operands that go through the unpack stage to form data inputs for all adder-supported operations. The multiplier/divider units send their intermediate results on the other path to the adder's input stage for final computation. No new instructions can enter the pipeline while the intermediate result travels from multiplier or divider to the adder for the cleanup cycles. The one data repacker in the FPU packs the final result produced by the adder to the correct data format.

We based the floating-point divide operation on the SRT divide algorithm,[8] which selects the quotient digit based on an estimation of the partial remainder. This technique has the advantage of not requiring a full-precision adder to add or subtract the partial remainder with a divisor multiple. Therefore it runs faster. The latency and repeat rates for floating-point divide operations are 23 and 22 cycles for single-precision operations and 36 and 35 cycles for double-precision operations. (See Table 1.)

The adder calculates square root by generating 1 root bit per cycle using the SRT algorithm. Since the adder also supports multiply and divide instructions, no new computational instruction may start while it calculates a square root. The square-root latency is 54 and 112 cycles for single- and double-precision operations.

Designers equipped the floating-point divider and the multiplier units with features that allow the circuit to power down at the end of every operation by recirculating zeros in the unit.

The floating-point register file is a 32-entry by 64-bit array with two read ports and two write ports. We dedicated one of the write ports for FP computational result writebacks and the other for FP load, store, and move instructions. In the case of reading and writing the same location in the same cycle, the register file locally bypasses the write data onto the read buses.

Stalls, slips, and exceptions

Pipeline hazards interrupt smooth pipeline flow (Figure 2), causing stalls, slips, or exceptions. In stall cycles, the pipeline does not advance. When the R4000 processes the stall, it restarts the pipeline and reissues several instructions to generate correct results.

For slips, such as the load interlocks detailed earlier, only the DF, DS, TC, and WB stages advance while the IF, IS, RF, and EX stages do not. When the slip condition is resolved, the instructions in the pipeline resume from whatever stage they are in. For exceptions, the processor suspends the normal sequence of instruction execution and transfers control to an exception handler, detailed later.

Figure 8 on the next page shows how the entire pipeline stalls for a data cache miss on load instruction 1. Since the load miss processing takes several cycles, the pipeline stalls until the secondary cache and main memory access completes. Note that before we got into the stall, instruction 4 may have used erroneous data in its EX stage that was bypassed from the load instruction. During the restart sequence, the processor repeats the EX stage for instruction 4 to obtain the correct data from the LOAD operation. The different stall types include

- *Data cache miss*, detected by the data tag check
- *Data first stage stalls*, which can occur for three mutually exclusive groups of instructions. 1) The pipeline stalls to resolve whether the FP instruction will cause an exception before moving on to guarantee precise exceptions. 2) The pipeline stalls to let the instruction sign extend the result. 3)The pipeline stalls to let the store buffer entries retire to memory because control logic has detected a load to the same memory location.
- *Instruction cache miss*, detected by the instruction tag check
- *Instruction translation look-aside buffer stalls*, for instruction TLB misses (explained in detail later)
- *Multiprocessor*, generated by requests from other processors

Slips occur when the result of an instruction is not available until the DS stage of an instruction, as occurs with loads. Floating-point instructions interlocked for resources also cause slips, as do integer instructions waiting for an integer multiply or divide operation to complete. Variable shifts and shifts greater than 32 bits also use slips since these operations take two cycles to complete.

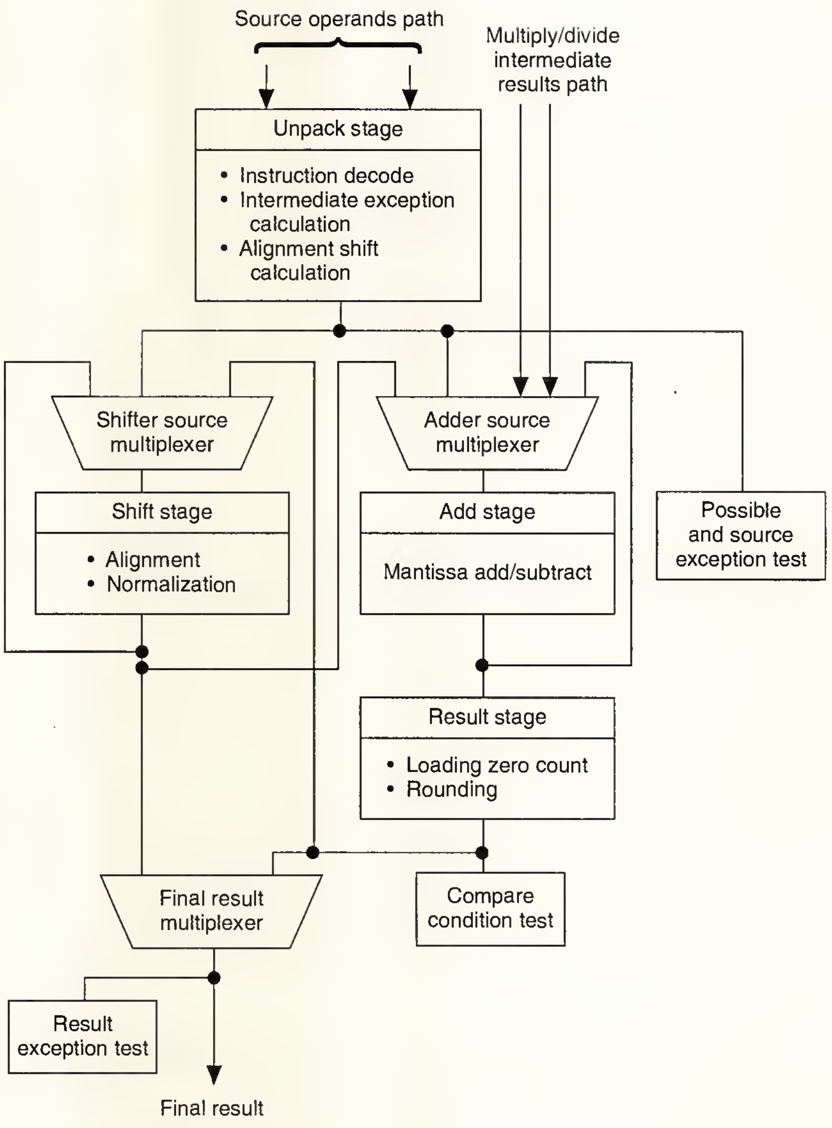


Figure 7. Adder logical block diagram.

Table 1. Integer and floating-point operation latencies and repeat rates in pipeline cycles.

	Integer		Floating point			
			Latency		Repeat	
	32 bits	64 bits	SP	DP	SP	DP
Add/subtract	1	1	4	4	3	3
Multiply	10	20	7	8	3	4
Divide	69	133	23	36	22	35

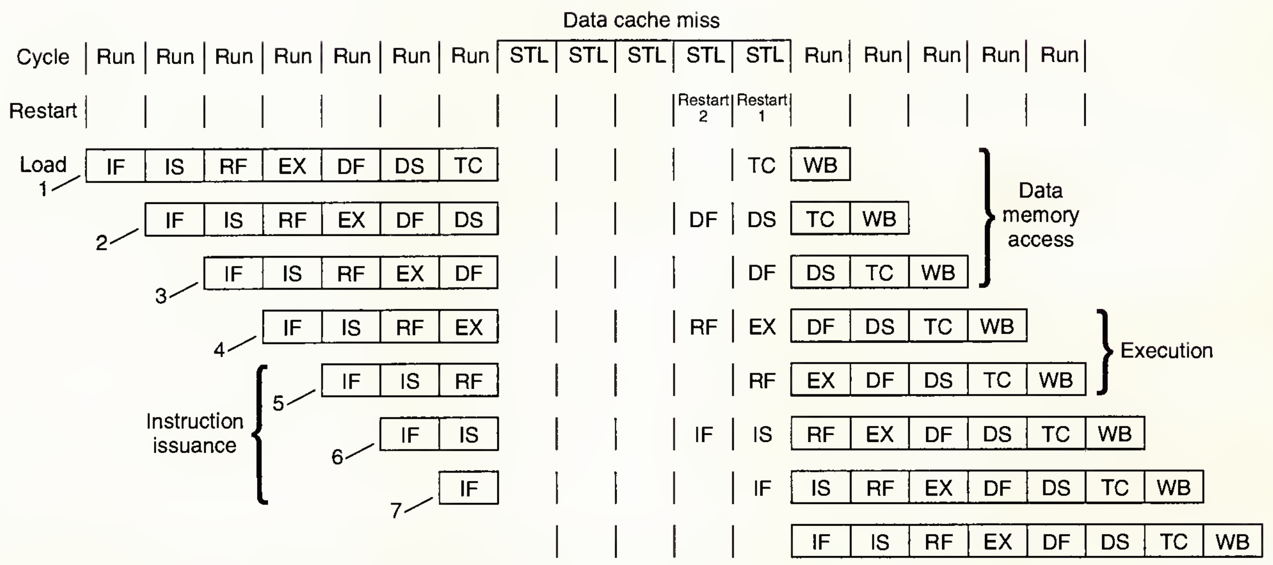


Figure 8. ADD data cache miss, use of load. STL indicates a stall.

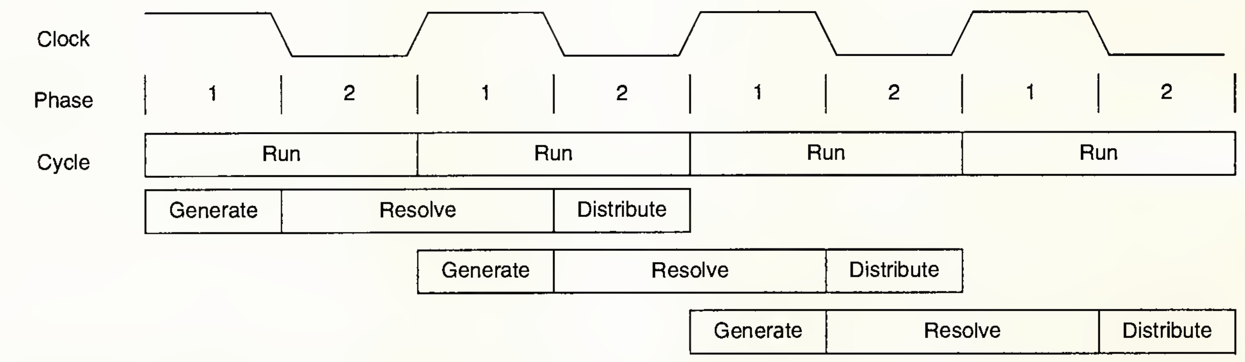


Figure 9. Circuit pipelining.

The R4000 processes many stalls and slips simultaneously. By slipping on instructions that need the same resources as a multicycle floating-point instruction, it can simultaneously accept other stall conditions from instructions that continue to advance further down the pipeline. Also, multiprocessor-initiated stalls, which can stall the pipeline to examine the cache, occur simultaneously with DCT, DFT, and ICT stalls described above.

Stall and slip implementation. The state machines that control pipeline flow (run, slip, and restart machines) operate in a pipelined fashion. When logic detects a stall or slip condition in a given cycle, the soonest the R4000 can process this condition is the end of the next cycle.

Figure 9 shows a sample timing diagram. In the first phase, the pipeline control unit evaluates logic that may generate a stall or slip condition. In phase 2 and the second phase 1, the state machines are resolved. Finally, the pipeline control signals are distributed throughout the chip during the second phase 2.

After processing a stall, the R4000 initiates a two-cycle restart sequence before the pipeline can run again. During this sequence, it reevaluates portions of the pipeline with corrected information before normal pipeline flow resumes. As shown in Figure 8, it repeats three activities: data memory access, execution, and instruction issuance.

Exception handling. The R4000 processes exceptions from sources in different pipeline stages. It prioritizes incoming exceptions and gives highest priority to the faulting instruction furthest along the pipeline. Table 2 lists different exceptions and the stages where they are signaled.

During normal processing, the R4000 nullifies pipeline stages for three reasons.

- When an exception occurs, it nullifies instructions after the faulting instruction.
- It nullifies certain instructions in branch delay slots when a branch is taken.
- When the pipeline slips, it creates a nullified instruction "bubble," as the back end of the pipeline advances and the front end does not.

After being nullified, the instruction does not commit to any state. For performance, the processor inhibits any stalls signalled by the instruction. For example, if an instruction will cause a data translation exception, which is detected at the end of the DS stage, the processor will not allow it to signal a cache miss in the TC stage.

Table 2. Exceptions.

Cycles	Exceptions
IF	–
IS	–
RF	Instruction translation
EX	Interrupt
	Bus error instruction
	Illegal instruction
	Breakpoint
	Syscall
	Coprocessor unusable
	ECC instruction
	Virtual coherency instruction
DF	–
DS	Overflow
	Floating point
TC	TLB modified
	Data translation
WB	Bus error data
	Virtual coherency data
	Watch
	NMI
	Reset

Memory management unit

The MMU translates virtual addresses into physical addresses using an on-chip translation look-aside buffer (TLB). It manages exceptions, controls the cache subsystem, and provides diagnostic and error recovery facilities. Compared to the R3000, the R4000 MMU provides enhanced operating system support including increased TLB entries, variable page sizes, 64-bit architecture support, supervisor privilege level, timer interrupts, and a physical address trap.

We wanted to increase the number of entries in the TLB over the 64 entries available in the R3000 since this boosts performance in a wide range of applications. Using 128 entries required too much area for the fully associative lookup circuit. Therefore, we implemented a 48-entry TLB with each entry mapping two consecutive pages and producing 96 effective entries. The TLB superpipelines in the R4000 (across the DF/DS pipeline stages) and runs in parallel with the cache access.

The instruction translation look-aside buffer (ITLB) is a two-entry, fully associative translation buffer that is a subset of the main TLB. This ITLB supports only a 4-Kbyte page size, to reduce complexity with minimum performance impact. When an instruction miss occurs in the instruction buffer, the pipeline stalls and the main TLB refills the ITLB. When a branch is taken into a different page, the branch target instruction address translation uses the TLB bandwidth available during the data first and data second stages of the branch instruction. Since the instruction first and instruction second stages of the branch target line up with the data first and data second stages of the branch instruction, the target address translation refills the ITLB without stalling the pipeline.

The R4000 implements variable page sizes on a per-page basis, varying from 4 Kbytes to 16 Mbytes. This helps to reduce thrashing of the TLB in some cases, such as in the use of a frame buffer which uses large data blocks. It implements variable page sizes by having a mask associated with each TLB entry. When addresses approach the TLB for translation, the corresponding mask bits in the TLB specify which virtual address bits participate in the comparison and translation.

The R4000 instruction set architecture supports 64-bit addressing. The current revision of the R4000 uses 40 bits of the 64-bit virtual address space. Increasing the effective virtual address size above 40 bits would have made the TLB wider than the data path and difficult to fit into the layout. Hardware explicitly checks the unused upper bits (bits 61:40) of the virtual address to make sure they are zero, ensuring a smooth transition for software as the size of the virtual address grows in future revisions. The R4000 supports a physical address of 36 bits.

The unit includes a supervisor privilege level of operation, in addition to the kernel and user levels present in previous company designs. This mode improves operating system support with more privilege levels.

A CACHE instruction provides a set of operations allowing the implementation of both a high-performance, symmetric, multiprocessing operating system and a high-performance workstation operating system. This instruction makes some tasks more efficient, including block copy, page zeroing, cache initialization, page flushing, and cache testing.

The CACHE instruction supports a number of operations including

- load and store of cache tags,
- selective invalidation of cache lines,
- create dirty exclusive data cache lines, and
- forced writeback of lines.

The R4000 provides a physical address trap feature for debugging software. This takes an exception on a reference to a selected physical address, which is specified in the Watch register.

The Count and Compare registers implement a timer interrupt service. The Count register acts as a timer, incrementing at half the pipeline clock rate. When the value in the Count register equals the value in the Compare register an interrupt occurs.

Memory hierarchy

The R4000 fits a range of system configurations. A programmable system interface permits tuning to different system specifications and exploiting future improvements in DRAM and SRAM design. The R4000 supports a two-level cache hierarchy that configures to run with different line sizes. Multiple cache coherency protocols available on the R4000 support several multiprocessor systems.[9,10]

The limited available primary cache size necessitated support for a closely coupled off-chip secondary cache required by high-end systems. We estimated the cache control section required 10 percent extra logic to support systems both with and without secondary cache. The R4000 manages its primary and secondary caches using a write-back method, in which stores send data into the caches, but the data do not write back to memory until the cache line is replaced or flushed.

The processor maintains its primary caches as a subset of the secondary cache contents. This prevents the occurrence of virtual aliases, which could lead to incorrect operation. A virtual alias occurs when multiple virtual addresses in the primary cache map to the same physical address in the secondary cache.

The primary caches are virtually indexed, so the secondary cache stores 3 bits of the virtual address (bits 14 to 12) needed to locate the primary cache lines that may contain data from a particular secondary cache line. (This virtual address information will support primary caches up to 32 Kbytes each). Because only one copy of the secondary cache line can reside in the primary cache, no two virtual addresses in the primary cache can map to the same physical location. Without this capability, R4000 would have to flush the large secondary cache to prevent aliasing. This is time consuming, especially for aliases caused by reusing pages for I/O.

Primary cache. While the initial version of R4000 uses an on-chip primary cache size of 8 Kbytes of instruction and 8 Kbytes of data, we can easily increase these sizes. The current revision supports primary caches up to 32 Kbytes each of instruction and data.

The primary cache is a direct-mapped, virtually indexed, physically tagged cache. Direct mapping makes it easy to find the location of a particular line in the cache and to manage cache consistency between the primary and secondary caches.

As the primary cache is virtually indexed, the virtual address generated by R4000's address unit looks up the cache line, while the address translation occurs in parallel. The address translation produces the physical address of the access, and the comparator compares it with the physical address read from the tag of the cache lines. The processor uses data coming out of the cache before it checks the tag, reducing the delay before load data can be used by one cycle.

Direct-mapped caches access faster than associative caches, but their hit rate is not as high as for set-associative caches. This penalty decreases as we increase the size of the primary caches. The primary caches support two software-programmable line sizes (16 and 32 bytes) that users can change independently for the instruction and data caches.

R4000 needs two cycles to access data in the primary cache, but a new address may enter every cycle. This is possible because the processor accesses the cache array in one cycle, excluding the address buffering and the data drive time. The address does not acces the array until the beginning of phase 2 of the first cycle, when the data from the previous access have been latched.

The primary instruction and data caches have separate data and tag arrays. The data cache data array and tag array may be addressed separately every cycle. During the data first and data second stages of a store instruction, the processor accesses the tag array for the store, while it may access the data array for a previous store that has passed its tag check and has data waiting in the store buffer. The two-entry store buffer decouples the data to be stored from the rest of the pipeline.

Since the architecture supports byte stores, the data cache array is arranged in eight blocks. Each block has a byte of data, a parity bit, and a redundant bit. The primary caches access 64 bits of data at a time, with the ability to write selected bytes. Row and column redundancy terms improve the die yield. To replace a defective row or column in one of the cache arrays with a redundant row or column, the manufacturer must blow the laser fuses.

Secondary cache. The secondary cache is direct mapped, physically indexed, and physically tagged. Manufacturers can build it from industry-standard static RAMs of different speeds and densities. The 128-bit-wide secondary cache interface allows a single access to the secondary cache to fill a four-word primary cache line. This cache supports a line size of four, eight, 16, or 32 words.

A physically indexed secondary cache makes multiprocessor support easy as all addresses on the system bus can be physical, eliminating the need for extra address translation information.

With R4000 supporting a maximum secondary cache size of 4 Mbytes, and with several such caches present in a multiprocessor system, the probability of a soft error demands support for error checking and correction. This ECC support for the secondary cache corrects 1-bit errors and detects 2-bit errors. R4000 performs on-chip tag correction, but it needs external hardware support to correct data errors.

We chose parity support for the primary cache since the on-chip caches are small and less prone to soft-error failure. If the operating system finds a parity error in the primary cache on a clean line, it can arrange to refill the primary cache line. When it detects a cache error, the processor takes an exception and jumps to uncached space. There the operating system examines the cache error control register, which specifies the type and location of the cache error.

One complex operation carried out in the cache logic is the write-back of dirty lines to memory. During writebacks, a state machine, the *zipper*, merges dirty (corrupted) lines in the primary cache with the data from the secondary cache as the line transfers to the system interface. The zipper checks tags in the primary instruction and data caches. It invalidates both instruction and data lines while merging any dirty data from the primary data cache. This operation completes in four pipeline cycles to match the maximum speed supported by the secondary cache.

System interface. The system interface lets the processor access external resources required to satisfy cache misses. It also allows an external agent access to some of the processor's internal resources. For multiprocessor systems, the system interface provides the processor mechanisms necessary to maintain cache coherence of shared data.

R4000 uses a 64-bit-wide system interface to increase main memory bandwidth compared with previous 32-bit system interfaces. The system interface can receive a double word every two pipeline cycles. If R4000 is operating without a secondary cache, the system interface can operate at the maximum system interface data rate, since the primary cache has a 64-bit data path that supports this rate. With a secondary cache, the maximum data rate the processor can support directly relates to the secondary cache access time. If the access takes too long, the processor cannot transmit or accept data at the maximum rate. The secondary cache only accepts reads and writes occurring in at least four cycles. With fast static RAMs that support a four-cycle access, the secondary cache interface can keep up with data coming in from the system interface at the maximum rate. Designers can program the system interface to transmit data in a range of rates, to suit different system and secondary cache speeds.

The system interface can be programmed to be clocked by a divided-down version (divided by two, three, or four) of the internal clock frequency. The internal clock runs at twice the processor's input, or master, clock. This allows systems designed for slower versions of the R4000 to run faster versions. For example, a system designed for a 50 MHz R4000 (with the system interface programmed to halve the internal 100 MHz pipeline clock) could implement a 75 MHz R4000 with the system interface clock divisor changed to divide by three. A 75 MHz external clock generates a 150 MHz internal pipeline clock, which the divisor divides by three to produce a 50 MHz system clock.

The R4000 supports an overlapped mode of operation on the system interface when configured with a secondary cache. When a miss occurs in the secondary cache that requires a line to be written back to main memory, the system interface sends out a read request for the miss and then immediately sends out a write with the writeback data. This saves the R4000 from having to buffer up secondary cache lines before they are written back, which would use significant chip area to support the largest secondary cache line of 32 words.

Multiprocessor support. The R4000 provides mechanisms to implement a variety of cache coherency protocols that may be snoopy or directory based (see Figure 10). Designers closely coupled the multiprocessor logic with the pipeline activity to allow access to the primary caches.

The starting point for R4000's coherency model was the MESI (modified, exclusive, shared, invalid) protocol. MESI implements a four-state cache coherence protocol (the states are invalid, clean exclusive, dirty exclusive, and shared). R4000 implements a fifth, the dirty shared state (Figure 11 on the next page), which allows for efficient implementations of a semaphore given the support for update protocol. When a processor successfully acquires a semaphore by gaining a dirty shared copy of the semaphore, all the other processors using that semaphore will be updated with its new value. They don't need to generate additional transactions on the bus. With the MESI protocol, a request from another processor (that is, an intervention) can cause writebacks to the sys-

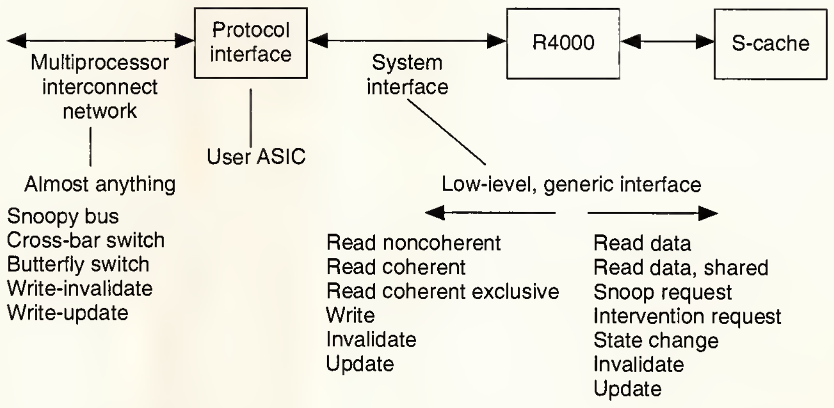


Figure 10. Multiprocessor protocols.

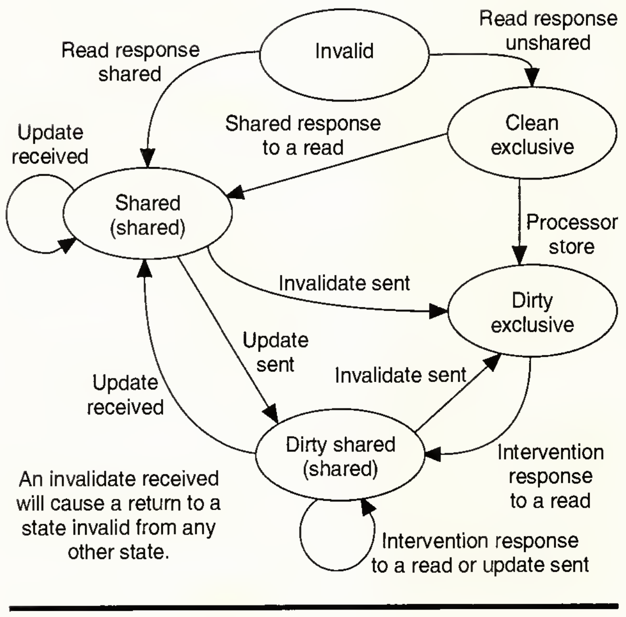


Figure 11. Cache coherency diagram.

tem memory. These writebacks place an additional burden on the system design. (The R4000 cannot process three-party transactions on interventions.) The processor stores the state of a cache line along with the tag and data for each line in the caches.

When R4000 receives an external snoop, intervention, invalidate, or update, it checks the secondary cache tag and state bits while allowing the processor to operate within the primary cache space in parallel. Misses in the secondary cache require no further action because the primary is a subset of the secondary. If an external event hits in the secondary cache, access to the primary may be required to complete the transaction. To gain access to the primary cache, the processor stalls the CPU pipeline.

The processor supports write-invalidate and write-update protocols, controlled on a per-page basis. The TLB may mark pages as uncached, noncoherent, coherent exclusive, coherent-write exclusive, and coherent-write update. Table 3 shows examples of the actions caused by these attributes.

Table 3. Examples of actions caused by coherency attributes.

Algorithm	Load-miss	Store-miss
Uncached	Word read	Word write
Noncoherent	Block read noncoherent	Block read noncoherent
Coherent exclusive	Block read exclusive	Block read exclusive
Coherent write exclusive	Block read	Block read exclusive
Coherent write updat	Block read	Block read/update

The R4000 provides a load linked and store conditional pair of instructions to provide synchronization between processors on the system bus based only on cache coherency. An example of this is the fetch-and-add operation.

```
Loop: ll     T0,0 (T1)   ;load counter, set load link bit
      addu   T0, T0, 1   ;increment
      sc     T0, 0 (T1)  ;store back if load link bit still set
      beq    T0, 0, Loop ;retry if store failed
```

The store conditional instruction fails if the location has been invalidated or updated since the preceding load linked instruction. This mechanism can implement semaphores, bitlocks, fetch-and-add, and other synchronization mechanisms. It also guarantees that at least one processor on the bus will get the semaphore on the first attempt so deadlocks or long stalls will not occur.

Design methodology

We chose full-custom data path layout for maximum speed and the highest packing density. Designers implemented most of the control sections using a logic synthesis and optimization tool and laid them out using standard cell place-and-route methodology. However, to achieve our target cycle times, we had to custom design and lay out by hand some of the control sections in the critical paths.

We used a two-phase, zero-overlap clock strategy and distributed it throughout the chip with a balanced clock tree, to control skew. A phase-locked loop generates four times the frequency of the external input (master) clock and distributes it through the chip. Divide-by modules at the end of the clock tree generate 2×- and 1× clocks. The processor pipeline and most logic use the 2× clock, which cycles twice as fast as the master clock frequency. The integer multiplier and floating-point multiplier use the high-speed 4× clock, four times the master clock frequency.

The chip uses two types of register/latches: stacked and pass-gate dynamic. Stacked registers, used extensively, are immune to clock skew as long as there are zero or an even number of inversions between the two stacked latches. However, when a short setup time and fast clock-to-output delay were necessary, we used pass-gate dynamic latches. In these cases, a design rule enforced a delay equivalent to the time needed to pass through at least three inverters of a fan-out of three between the latches to prevent data slip-through.

We equipped the output buffers with a digitally controlled slew rate to reduce noise injected into the system buses. One buffer determines the digital control signal values for the rest of the buffers. This output buffer sends the pad a signal, which in turn feeds back into an input pad. The processor samples the round-trip delay and references it with the clock cycle. Users can program the desired amount of skew in terms of a fraction of a clock cycle. Depending on the control signals generated, the strength of the output buffer's pullup and pull downs are adjusted. (See Figure 12.)

We laid out the chip using a generic Mips design rule, so all our semiconductor partners can work from a single database. This database is based on a 1.0-μm-drawn, two-layer metal, CMOS technology. Manufacturers are producing the R4000 in 0.8μ technology.

Verification

Mips carried out a functional simulation of a register transfer level model during the development of the R4000. The RTL model executes at about 1,000 processor cycles per minute on a 20-MIPS, R3000-based Magnum workstation. Designers divided the chip into major functional blocks (CPU, FPU, MMU, caches, and system interface) and wrote directed diagnostic tests to exercise these functional units. Trace comparisons of diagnostic tests run on an instruction-level simulator and on the R4000 RTL verified compliance with our architecture. To trace all the required signals and data in the R4000 superpipeline, we added more verification logic to the R4000 RTL model so it could capture traces for comparison with the instruction-level simulator traces.

We perfomed extensive automatically generated random diagnostic tests, again using our instruction-level simulator for trace comparison. We wrote additional verification diagnostics to ensure that all the arcs of the state machines within the R4000 were exercised. Our designers executed R4000 diagnostics within an RTL model of a system configurable at runtime to include a secondary cache and change any of the programmable parameters that control the system interface. They booted the Unix operating system on the R4000 RTL model about six months before Mips gave the design to its manufacturing partners. It took a 50-MIPS Mips 6280 seven days of processing to reach the Unix prompt.

We verified the multiprocessor capabilities of the R4000 using a number of different simulation models. A uniprocessor RTL simulation of the R4000 checked that the R4000 could generate and process all the multiprocessor requests defined by the R4000 interface specification. We also developed a simulation environment that could support multiple R4000 processors at the RTL level. Under this environment we ran directed diagnostic tests and self-checking random tests.

Finally, we verified that the physical implementation of the R4000 matched the RTL description by generating a gate-level model from schematics. Obviously, this model ran much slower than the RTL model, and so we needed a large compute resource to run the diagnostic test suite at the gate level. In the final stages of verification we used ten 6280 machines and around thirty 20-MIPS Magnum workstations.

Testability and packaging

The R4000 implements JTAG (IEEE Std. 1149.1) boundary scan specifications, intended to provide a test capability for the interconnection between the R4000 processor, the printed circuit board, and other components on the board.

The chip comes in two package configurations. The R4000MC and R4000SC, which have the 128-bit data interface to the secondary cache, are packaged in a 447-pin lead or plastic grid array. The R4000MC supports multiprocessor systems while the R4000SC supports high-performance uniprocessor systems. The R4000PC, for desktop, low-end servers, and embedded control systems, comes in a 179-pin PGA with no secondary cache interface.

TABLE 4 LISTS SPECMARKS FOR SIMULATED RESULTS of a realistic memory system. (See next page). We simulated the CPU time and most of the important aspects of memory and heuristically added the I/O times. Correlation of simulations with R4000 systems in the lab show the simulations to be pessimistic.

Acknowledgments

The R4000 became a reality due to an enormous team effort managed by our leader, Tom Riordan. We also had

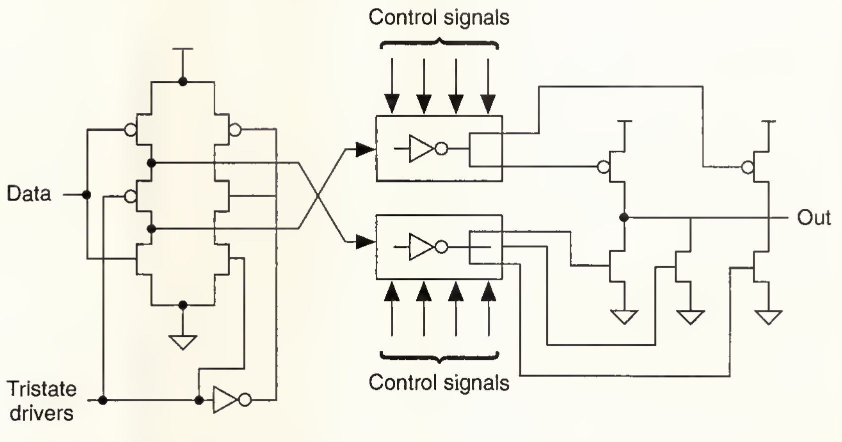


Figure 12. Output buffer.

Table 4. Simulated Specmarks for a 50-MHz external-clock R4000.

Benchmark	S-cache size 4 Mbytes	S-cache size 512 Kbytes	P-cache only
Gcc	46	43	27
Espresso	54	54	38
Spice2g6	42	38	27
Doduc	49	46	33
Nasa7	56	46	43
Li	66	65	47
Eqntott	54	52	50
Matrix300	278	273	177
Fpppp	55	54	29
Tomcatv	58	59	37
Simulated SPEC	63	59	42
Simulated SPEC int	55	53	39
Simulated SPEC fp	69	64	44
CPI (simulated SPEC)	1.5	1.6	2.3

great support from other groups within Mips that had to put up with our constant demands for support. Unfortunately, we cannot list each member of the R4000 team, but we thank them all.

References

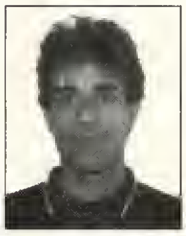
1. J. Hennessy et al., "Mips: A VLSI Processor Architecture," Tech. Report 223, Computer Systems Laboratory, Stanford University, Stanford, Calif., 1983.
2. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, Calif., 1990.
3. J.R. Mashey, "64-Bit Computing," *Byte*, Sept. 1991, pp. 135-142.
4. *MIPS R4000 Microprocessor User's Manual*, Mips Computer Systems Inc., Sunnyvale, Calif., 1991.
5. S. Waser and M. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart, and Winston, New York, 1982.
6. *ANSI/IEEE Std. 745-1985, Standard for Binary Floating-point Arithmetic*, IEEE, New York, 1985.
7. C. Rowen, M. Johnson, and P. Ries, "The Mips R3010 Floating-Point Coprocessor," *IEEE Micro*, Vol. 8, No. 3, June 1988, pp. 53-62.
8. D.E. Atkins, "High-Radix Division Using Estimates of the Divisor and Partial Reminders," *IEEE Trans. Computers*, Vol. C-17, No. 10, Oct. 1968, pp. 925-934.
9. S.A. Przybylski, *Cache and Memory Hierarchy Design*, Morgan Kaufmann, 1990.
10. A.J. Smith, "Cache Memories," *Computing Surveys*, Vol 14, No. 3, Sept. 1982.

Sunil S. Mirapuri defines and designs advanced microprocessor products at Mips Computer Systems. Previously, he worked for Rolm Milspec Computers and Intel Japan. His research interests include computer architecture, digital systems, and computer programming.

Mirapuri received his BS and MS degrees in electrical engineering from Stanford University. He is a member of the IEEE Computer Society.

Michael S. Woodacre is a member of Mips' VLSI design verification team. Prior to joining Mips to work on the R4000, he was a VLSI design engineer for Inmos' transputer microprocessor team.

Woodacre received a BS in computer systems engineering from the University of Kent in Canterbury, England.



Nader Vasseghi is a member of Mips' VLSI design group. He worked on the design and development of the R4000 and now works on the next-generation RISC processor. Prior to joining Mips, he designed network controller products and high-speed programmable array logic devices at Advanced Micro Devices.

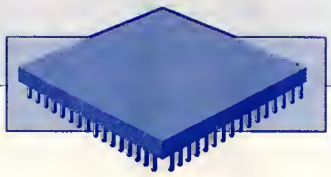
Vasseghi received his BSEE from the University of California at Santa Barbara and his MSEE from Southampton University in England. He is a member of the IEEE Computer Society.

Address questions regarding this article to Sunil Mirapuri at Mips Computer Systems, 928 Arques Ave., Sunnyvale, CA 94086; or via e-mail at sunil@mips.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 153	Medium 154	High 155



The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms

The Message-Driven Processor, an integrated multicomputer node, provides efficient mechanisms for parallel computing. Rather than being specialized for a single model of computation, the MDP incorporates primitive mechanisms for communication, synchronization, and naming. These mechanisms efficiently support most proposed parallel programming models. Each processing node of MIT's J-Machine consists of an MDP with 1 Mbyte of DRAM. MDPs have been operational since June 1991, and J-Machines built from them went on line in July 1991.

William J. Dally

J.A. Stuart Fiske

John S. Keen

Richard A. Lethin

Michael D. Noakes

Peter R. Nuth

Massachusetts Institute of Technology

Roy E. Davison

Davison Design and Development Corp.

Gregory A. Fyler

Intel Corporation

The Message-Driven Processor is a 36-bit, 1.1-million transistor, VLSI microcomputer specialized to operate efficiently in a multicomputer. The MDP chip includes a processor, a 4,096-word by 36-bit memory, and a network port. An on-chip memory controller with error checking and correction (ECC) permits local memory to be expanded to one million words by adding external DRAM chips.

The processor is message-driven in the sense that it processes in response to messages, via the dispatch mechanism. No receive instruction is needed. The MDP creates a task to handle each arriving message. Messages carrying these tasks advance, or drive, each computation.

We designed the MDP with two primary goals in mind.

- We wanted to implement a general-purpose, multicomputer processing node that provides the communication, synchronization, and naming mechanisms required to efficiently support several different parallel programming models.
- We wanted to create an inexpensive, VLSI component for cost-efficient parallel computers. Ideal nodes should be inexpensive and plentiful VLSI commodity parts—as inexpensive and plentiful as jellybean candies—that can network together to form a Jellybean Machine (J-Machine) multicomputer.

Efficient parallel mechanisms

Computer hardware provides primitive operations called mechanisms. These mechanisms build the abstractions that in turn make up a programming system.[1] For example, most sequential machines provide some mechanism for a push-down stack to support the last-in-first-out (LIFO) storage allocation required by many sequential programming models. Most machines also provide some form of memory relocation and protection to allow several processes to coexist in memory at once without interference. The proper set of mechanisms can significantly improve performance over a brute-force interpretation of a programming model.

Over the past 40 years, sequential von Neumann processors have evolved a set of mechanisms appropriate for supporting most sequential programming models. It is clear, however, from efforts to build concurrent machines by wiring together many sequential processors, that these highly evolved sequential mechanisms do not adequately support most parallel models of computation. These mechanisms do not efficiently support synchronization of events, communication of data, or global naming of objects. As a result, designers must implement these functions, inherent to any parallel model of computation, largely in software with prohibitive overhead. For example, sequential machines require hundreds of instructions to create a new process. This cost prohibits the use of fine-grain programming models where processes typically last only a few tens of instructions.

The MDP supports a broad range of parallel programming models, including shared-memory,[2] data parallel,[3] dataflow,[4] actors,[5] and explicit message-passing,[6] by providing low-overhead primitive mechanisms for communication, synchronization, and naming. Its communication mechanisms permit a user-level task on one node to send a message to any other node in a 4,096-node machine in less than 2 µs. This process doesn't consume any processing resources on intermediate nodes, and it automatically allocates buffer memory on the receiving node. On message arrival, the receiving node creates and dispatches a task in less than 1 µs.

Presence tags provide synchronization on all storage locations. Three separate register sets allow fast task switching. A translation mechanism maintains bindings between arbitrary names and values, and supports a global virtual address space. We selected these mechanisms to be both general and amenable to efficient hardware implementation. To support fine-grain, concurrent programming systems, we designed the mechanisms to efficiently handle small objects (eight words) and small tasks (20 instructions).

3D array of fine-grain, processing nodes

The MDP is an example of an inexpensive, fine-grain, multicomputer building block. A fine-grain node does not necessarily have a slow processor. We can build a competent processor in a fraction of a modern VLSI chip's area. Fine grain and small memory decrease the chip's cost, resulting in greater arithmetic performance and local memory bandwidth per unit cost. Fast communication and a global address space prevent the small local memories from limiting programmability or performance.

In a multicomputer, system cost is very sensitive to processor cost. A less-expensive node results in a comparably priced system with more processors and, to first order, higher performance. In these systems, designers avoid costly features that give a small incremental return in processor performance (such as large caches) in favor of building systems with more nodes, an option not available to the designer of a sequential computer.

The 3D network that connects MDPs gives the highest throughput and lowest latency for a given wire density.[7] This network allows the processing nodes to be packed densely and results in uniformly short wires. It does not waste communication bandwidth by embedding an esoteric topology into physical space. Messages traveling through the network follow a Manhattan shortest path in physical space; they never backtrack. (A Manhattan path travels forward, to the side, and up or down, but not across diagonals.)

Background

The MDP builds on previous work in multicomputer design. Like the Caltech Cosmic Cube,[6] Intel's iPSC,[8] the Ncube,[9] and the Ametek,[10] each MDP in the J-Machine has a local memory and communicates with other nodes by passing messages. Because of its low overhead, the MDP can exploit concurrency at a much finer grain than these early message-passing multicomputers. Delivering a message and dispatching a task in response to the message's arrival takes less than 2 µs on the J-Machine, as opposed to 5 ms on an iPSC-1 or 300 µs on an iPSC-2.

Like the BBN Butterfly[11] and the IBM RP3,[12] the MDP supports a global virtual address space. The same IDs (virtual addresses) reference local (on the same node) and remote (on a different node) objects. Like the Inmos transputer,[13] the Caltech Mosaic,[14] and the Intel iWarp,[15] the MDP is a single-chip processing element integrating a processor, memory, and a communication unit. The MDP is unique because it extends these previous efforts with efficient primitive mechanisms for communication, synchronization, and naming.[1] It uses a direct communication network based on work reported by Dally,[7] Dally and Seitz,[16] and Dally and Song.[17]

System architecture

To the hardware designer, the MDP appears as a component with a memory port, six two-way network ports, and a diagnostic port, as shown in Figure 1.

The memory port provides a direct (that is, no glue) interface to up to 1 Mwords of ECC DRAM, consisting of 11 multiplexed address lines, a 12-bit data bus, and three control signals. Static-column or page mode DRAMs cycle three times to access a 36-bit data word and a fourth time to check or update the ECC check bits. Current J-Machines use three 1M × 4 memory parts to form a four-chip processing node with

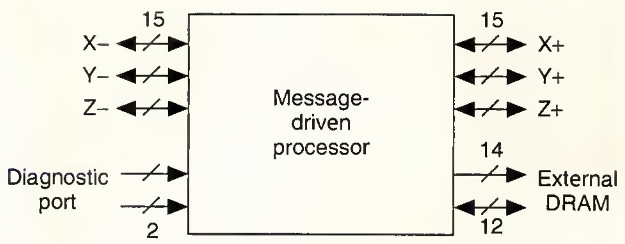


Figure 1. MDP pinout. The MDP has a memory port (26 pins), six network ports (15 pins each), and a diagnostic port (three pins).

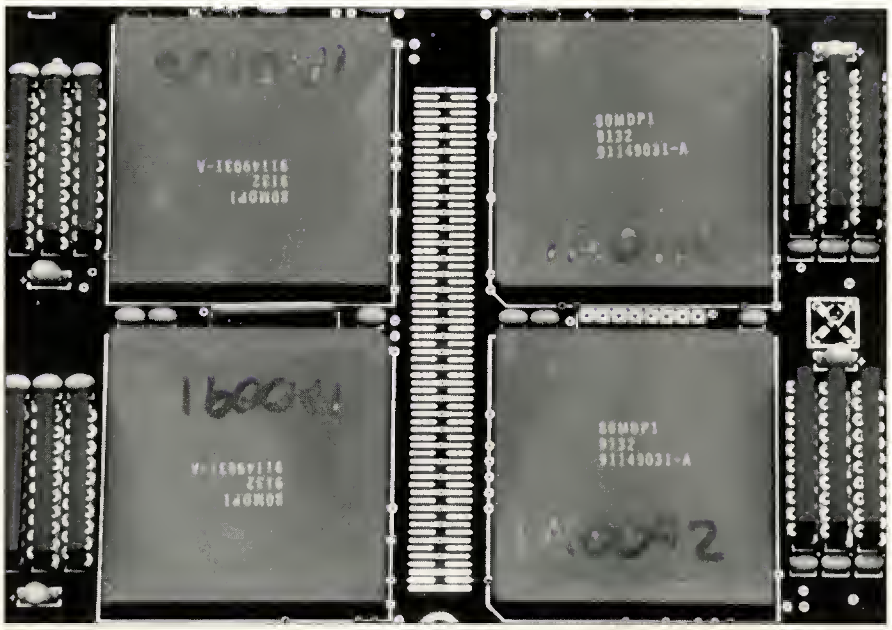


Figure 2. An array of four J-Machine processing nodes. Each node consists of one MDP chip and three 1M × 4 static-column DRAMs. With conventional packaging the node measures 2 in. × 2.75 in.

262,144 words of memory that measures 2 in. × 2.75 in., as shown in Figure 2.

The network ports connect MDPs together in a 3D mesh network. Each of the six network ports corresponds to one of the six cardinal directions (*+X,–X,+Y,–Y,+Z,–Z*) and consists of nine data and six control lines. Each port connects directly to the opposite port on an adjacent MDP. We give details of the 3D network later in this article.

The diagnostic port issues supervisory commands and reads and writes MDP memory from a console processor. The port consists of two control lines, a serial input line, and a serial output line. Using this port, a console processor can read or write any location in the MDP's address space, as well as reset, interrupt, halt, or single-step the processor.

Software. To a systems programmer, a bare J-Machine appears as a collection of node memories and register files operable by an instruction set that includes communication, synchronization, and naming mechanisms. The systems programmer uses these mechanisms to implement a programming model. For example, one can build a shared memory model that gives the application programmer a single, shared address space.

The implementation of a combining tree[18] illustrates the use of the MDP mechanisms. The combining tree (Figure 3) consists of a number of nodes each containing a value, a count, and a pointer to a parent node.

We initialize the value to zero and the count to the number of inputs expected. To sum the values of a number of parallel processes, each node sends a COMBINE message containing the result of its process to a combining node. When the messages arrive, the processor containing the combining node creates a task to execute the COMBINE routine. The routine adds the message value to the node's value and decrements the count. When the count reaches zero, the node sends a COMBINE message to the node's parent.

Communication. The MDP supports communication using a SEND instruction for message formatting, a fast network for delivery, automatic message buffering, and task creation upon message arrival.

A series of SEND instructions carries a message of arbitrary length to any node in the machine. Upon arrival at the receiving node, a hardware queue buffers the message. When the message reaches the head of the queue, the node dispatches a task to handle the message. The combining tree example uses a pair of SEND instructions to send the COMBINE message to a node. Upon message arrival, the MDP buffers the message and creates a task to execute the COMBINE routine.

Synchronization. The MDP synchronizes using message dispatch and presence tags on all states. Because each message arrival dispatches a process, messages can signal events on remote nodes. For example, in the combining tree ex-

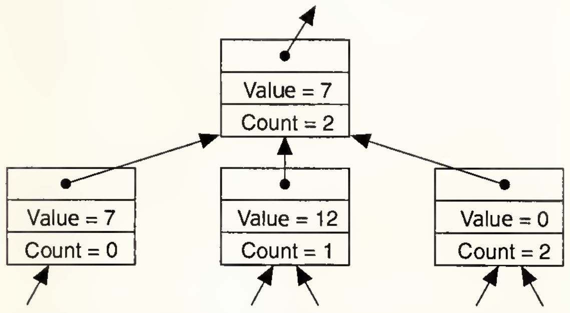


Figure 3. A combining tree sums results produced by a distributed computation. Each node sums the input values as they arrive and then passes a result message to its parent.

ample, each COMBINE message signals its own arrival and initiates the COMBINE routine.

In response to an arriving message, the processor may set presence tags for task synchronization. For example, access to the value produced by the combining tree may be synchronized by initially tagging as empty the location that will hold this value. An attempt to read this location before the combining tree had written it would raise an exception and suspend the reading task until the root of the tree writes the value. Synchronization on data availability in this manner is quite common in many parallel programs.

Naming. The MDP supports naming with segmented memory management and translation instructions. In the combining tree example, the MDP allocates a memory segment to hold the state of each combining node. Using a segment descriptor, it relocates and protects accesses to the node. To make combining nodes relocatable across processing nodes, the MDP translates a node's virtual address to find the processing node where it resides. Upon reaching this node, a second translation locates the segment descriptor for the combining node.

Instruction set architecture

The MDP extends a conventional microprocessor instruction set architecture (ISA) with instructions to support parallel processing. Specifically, the MDP provides efficient hardware mechanisms for communication, synchronization, and naming. Although we describe here the MDP ISA, with particular emphasis on these mechanisms, readers can find more details in Dally et al.[19]

Register set. The MDP provides separate register sets to support rapid switching between three execution levels: background, priority 0 (P0), and priority 1 (P1). The MDP executes at the background level when no messages are pending. Each arriving message creates a task and initiates execution at P0 or P1, depending on the message's priority. The MDP executes the highest priority task at any point in time. The arrival of a P1 message while the MDP is executing a P0 task causes the MDP to switch execution levels (and thus register sets). When the P1 task completes, the MDP resumes execution at P0 by switching to the P0 register set that holds the register state of the suspended task.

The register set at each priority level includes

- four general-purpose data registers, R0-R3,
- four address registers, A0-A3,
- four ID registers, ID0-ID3, and
- one instruction pointer, IP.

The background register set does not include ID registers. They only exist at P0 and P1.

Most instructions operate on the general registers R0-R3. Each address register A0-A3 contains a segment descriptor consisting of a base and a length field. Memory addresses are specified by an offset and an address register. For example, the operands [R0, A1] and [3, A2] specify an indexed access to the segment described by A1 and a displacement of three words into A2's segment.

ID registers usually hold object IDs. The instruction pointer includes process status bits that control virtual addressing, type checking, and fault handling. Placing these bits in the instruction pointer enables control and execution states to change by loading a single register. The relatively small size of each register set facilitates quick task switching within an execution level.

Tags. The MDP uses tags for type checking and synchronization. Every 36-bit word of register and memory state holds a 32-bit value and a 4-bit tag that indicates the type of the value. Tag values are defined for primitive user data types (such as symbol, integer, and Boolean) and for system data types, such as IP, Addr (a segment descriptor), and Msg (a message header). Four tag values are user-definable. If type checking is enabled, the MDP checks operand tags to determine which form of an instruction to execute. It raises an exception if the operands are incompatible with the instruction.

Two tags, Fut and Cfut, support intertask synchronization. A Cfut tag initially marks a location empty. When a task produces the value for the location, it overwrites the Cfut with the final value and tag. Any attempt to read from the location before the value is produced invokes the Cfut fault handler, which typically suspends the reading task until the location is written. Fut is used for global synchronization, and Cfut for local.

Hardware support for tags makes software more efficient and robust. A program can perform an operation without checking whether operands are present or of the correct type. For normal cases in which no fault occurs, execution proceeds faster than if special test and branch instructions were required to check for type and presence. Only exceptional cases incur the overhead of running a fault handler.

Instructions. The MDP executes 17-bit, fixed-format, three-address instructions with the format shown in Figure 4. Each instruction specifies an operation, two general register operands, and a third operand that may be a register, a memory location, or a constant. Two 17-bit instructions fit into each 36-bit word. Any instruction stream word not tagged as an

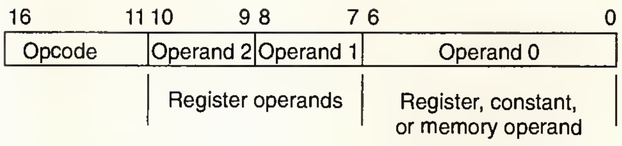


Figure 4. MDP instruction format.

General movement and type instructions

READ	WRITE	READR	WRITER	RTAG
WTAG	LDIP	LDIPR	CHECK	

Arithmetic and logic instructions

CARRY	ADD	SUB	MULH	MUL
ASH	LSH	ROT	AND	OR
XOR	FFB	NOT	NEG	LT
LE	GE	GT	EQUAL	NEQUAL
EQ	NEQ			

Network instructions

SEND	SENDE	SEND2	SEND2E

Associative lookup table instructions

XLATE	ENTER	PROBE

Special instructions

NOP	INVAL	SUSPEND	CALL

Branches

BR	BNIL	BNNIL	BF	BT
BZ	BNZ			

Figure 5. Six categories of MDP instructions.

instruction is loaded as a constant into register R0. This provides a very efficient means to load arbitrary 36-bit constants. Figure 5 summarizes the MDP instruction set by category.

Naming. The MDP supports naming via translation instructions and segmented addressing. Addressing memory through segment descriptors permits arbitrary size objects to be relocated and protected. The ENTER instruction enters an arbitrary translation from a 36-bit key to a 36-bit data value in a set-associative cache (translation table) mapped into the on-chip memory. The XLATE instruction looks up the data value (if any) associated with a key. These instructions can translate an object's name into a physical segment descriptor or a node number to support a global virtual address space.

Communication. The MDP provides hardware support for end-to-end message delivery including formatting, injection, delivery, buffer allocation, buffering, and task scheduling.

An MDP transmits a message using a series of SEND instructions, each of which injects one or two words into the network at either priority 0 or 1. Figure 6 shows a typical message send. The first SEND instruction reads the absolute address of the destination node in $<X,Y,Z>$ format from R0 and forwards it to the network hardware. The SEND2 instruction reads the first two words of the message out of registers R1 and R2 and enqueues them for transmission. The final instruction enqueues two additional words of data, one from R3, and one from memory. The use of the SEND2E instruction marks the end of the message and causes it to be transmitted into the network. This sequence executes in four clock cycles (250 ns).

The network delivers an injected message to the destination node, as described later. At the destination, a hardware-managed, FIFO queue in the internal RAM of the MDP buffers the message. Separate queues exist for P0 and P1 messages.

Task scheduling. When a message reaches the head of the highest priority nonempty queue, the MDP creates a task to handle it by changing the thread of control and creating a new addressing environment, as shown in Figure 7. Every message header contains a message opcode and the message length. The MDP loads the message opcode into the instruction pointer to start a new thread of control. The length field and the queue head create a message segment descriptor (automatically written to A3) that represents the initial addressing environment for the task. The message handler code may open additional segments by translating object IDs in the message into segment descriptors. Creating a task to handle a message takes three cycles.

The dispatch mechanism directly processes messages requiring low latency (for example, combining and forwarding). Other messages, such as a remote procedure call, specify a handler that locates the required method (using the translation mechanism described earlier) and then transfers control to the method.

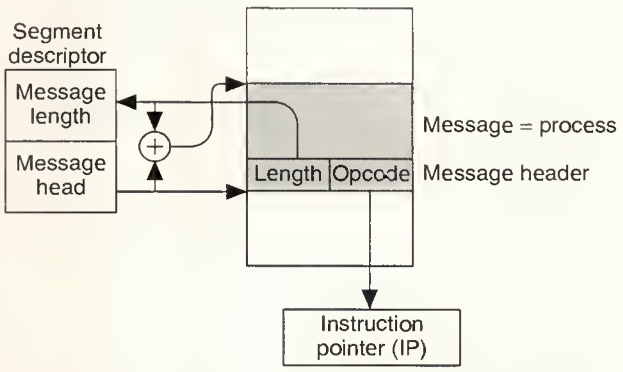


Figure 7. Message dispatch. In three clock cycles, a node creates a new task by setting the instruction pointer to change the thread of control and creating a message segment to provide the initial addressing environment.

```
SEND    R0,0          ; send net address (priority 0)
SEND2   R1,R2,0       ; header and receiver (priority 0)
SEND2E  R3,[3,A3],0   ; selector and continuation -
                        end msg. (priority 0)
```

Figure 6. MDP assembly code to send a four-word message uses three variants of the SEND instruction.

```
MOVE    [1,A3],R0     ; get method ID
XLATE   R0,A0         ; translate to segment descriptor
LDIP    INITIAL_IP    ; load instruction pointer to
                        transfer control to method
```

Figure 8. MDP assembly code for the CALL message.

For example, Figure 8 shows the CALL handler code handling a remote procedure call. Figure 9 depicts the execution of the handler. The first instruction gets the method ID (offset one word into the message segment referenced by A3). The next instruction translates this method ID into a segment descriptor for the method and places this descriptor in A0. In one of its operating modes, the MDP can use A0 as a pointer to a segment of code and IP as an index into that segment. This allows code to be easily relocated at runtime. The final instruction of the CALL handler transfers control to the method by loading the IP with a short integer offset. Thereafter the MDP will fetch instructions from the called method.

The method code may then read in arguments from the message queue. The XLATE instruction translates argument object identifiers to physical memory base/length pairs. If the method needs space to store local state, it may create a context object. When the method finishes executing, or when it needs to wait for a reply, it executes a SUSPEND instruction, which dequeues its message and passes control to the next message in the queue.

An example of a direct message handler is the COMBINE routine shown in Figure 3. Figure 10 displays the code for this routine. If the node is idle, execution of this routine begins three cycles after message arrival. The routine loads the combining node pointer and value from the message, performs the required add and decrement, and, if Count reaches zero, sends a message to its parent.

This 12-instruction routine executes in 21 cycles. It demonstrates several ways in which the MDP's communication mechanism reduces the overhead of message passing to the point where it can perform simple operations, such as combining. These ways include the following:

- The MDP hardware dispatches the COMBINE task by setting the instruction pointer to COMBINE and initializing message pointer A3 to allow direct access to message words. This avoids the overhead otherwise associated with control transfer and with setting up an addressing environment.
- The two SEND instructions transmit the four-word mes-

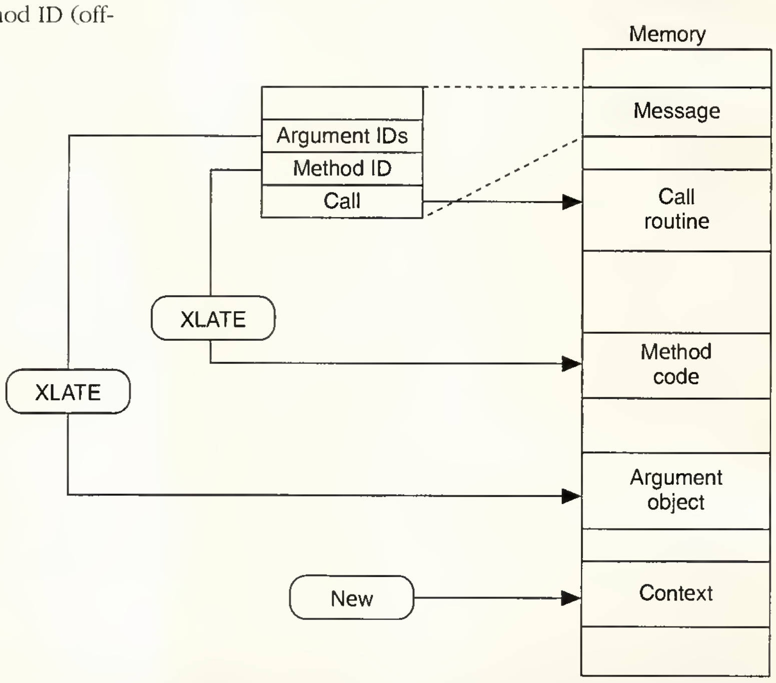


Figure 9. The CALL message invokes a method by translating the method identifier to find the code, creating a context (if necessary) to hold local state, and translating argument identifiers to locate arguments.

```
COMBINE:  MOVE     [1,A3], COMB              ; get node pointer from msg
          MOVE     [2,A3], R1                ; get value from msg
          ADD      R1, COMB.VALUE, R1
          MOVE     R1, COMB.VALUE            ; store result
          MOVE     COMB.COUNT, R2            ; get Count
          ADD      R2, -1, R2
          MOVE     R2, COMB.COUNT            ; store decremented Count
          BNZ      R2, DONE
          MOVE     HEADER,R0                 ; get message header
          SEND2    COMB.PARENT_NODE, R0      ; send message to parent
          SEND2E   COMB.PARENT, R1           ; with value
DONE:     SUSPEND
```

Figure 10. MDP assembly code for the combining tree example.

sage to the parent task. The message transmits directly from register and memory variables with no need to first format it in memory.

- The SUSPEND instruction terminates the task and simultaneously dequeues the message. If another message is pending in the queue, the processor dispatches a task to handle it two cycles after the execution of the SUSPEND instruction.

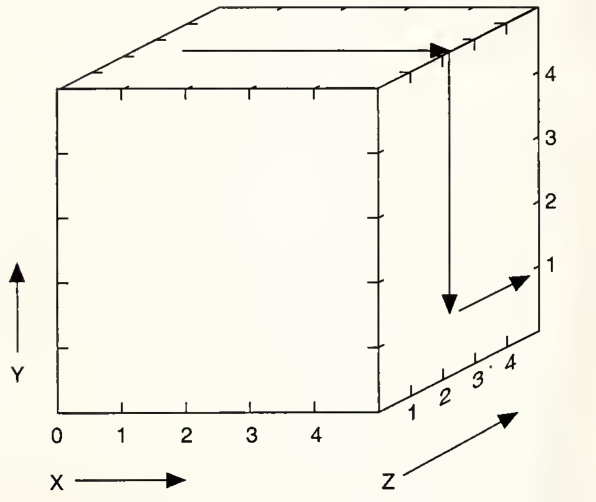


Figure 11. The J-Machine network is a 3D mesh or *k*-ary 3-cube. The network performs e-cube or destination tag routing. Messages route in each dimension in turn to the proper coordinate in that dimension. In this figure, a message routes from (1,5,2) to (5,1,4), routing first in *X*, then *Y*, then *Z*.

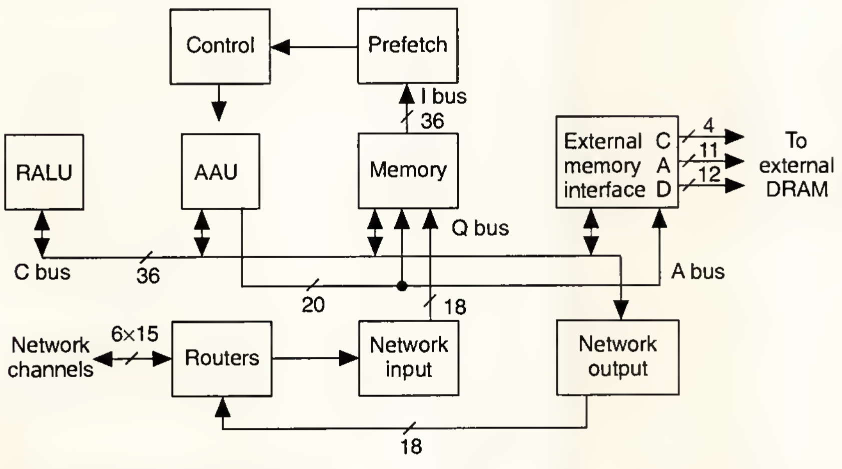


Figure 12. MDP block diagram.

Network architecture

The MDP contains a network interface and a router that support a communication network closely integrated with the processor. In a J-Machine composed of MDPs, the network provides end-to-end message delivery with low latency (less than 2 μs in a 4,096-node network) and high bandwidth (288 Mbits per second per channel). Message delivery occurs entirely within the routers of the machine and consumes no processor or memory resources at intermediate nodes.

Structure. The J-Machine network is a 3D grid, with two-way channels, dimension-order routing, and blocking flow control. (See Figure 11.) Addressing limits the size of the network to 65,536 nodes (32 × 32 × 64). Our initial prototype is a 1,024-node machine (8 × 8 × 16). The faces of the network cube are open for use as I/O ports to the machine. Each channel can sustain a data rate of 288 Mbps. All three dimensions may operate simultaneously for an aggregate data rate of 864 Mbps per node.

Three modules, shown in Figure 12, compose the network logic. The network output module buffers words and injects them into the network. The three routers, one for each dimension of the network, route messages from node to node. The network input module reassembles messages at their destination and buffers them into a message queue. We describe more details of implementation in the next section.

Engineering. We chose the 3D mesh topology of the J-Machine network as the most efficient arrangement subject to constraints of wiring density and component pinout.[7] These constraints set the width of the six bidirectional channels per MDP node at 9 data bits plus 6 control bits. We built the J-Machine as a stack of boards with dense board-to-board interconnections to implement the 3D network with short wires.

The MDP breaks with the tradition of asynchronous network routers by implementing a synchronous router.[16,17] This router operates at twice the rate of the processor, sending a pair of 9-bit *phits* between nodes each 62.5-ns processor cycle (A phit is a physical digit, the width of the physical channel. A pair of phits form a *flit*, or flow-control digit, the granularity of flow control in the network. An 18-bit flit is half an MDP data word.)

Each of the six bidirectional channels can be turned around on alternate cycles with no contention penalty. A novel pad design tolerates clock skew between routers and eliminates the potential for conduction overlap when the channel reverses direction.[20] Messages route through the network with a latency of one 62.5-ns processor cycle per hop. Thus, message latency *T* is given by

$$T = T_c(2L + D),$$

where T_c is the processor cycle time, L is message length in words, and D is the distance (number of nodes) a message must traverse. For example, in a 1,024-node machine, an L=6 word message to a random destination traverses an average of D=10 nodes for a latency of T=22 cycles or 1.4 μs. The bisection bandwidth (the bandwidth across a plane dividing the machine into two equal halves) of a 1,024-node machine is 18.4 Gbps. The aggregate bandwidth of the network channels is 864 Gbps, and the I/O bandwidth is 184 Gbps.

Routing and flow control. The J-Machine uses deterministic dimension order routing, also called e-cube routing. As shown in Figure 11, all messages route first in the X dimension, then in Y, then in Z. Since messages route in dimension order and messages running in opposite directions along the same dimension do not block, we avoid resource cycles, and leave the network provably deadlock free.[21]

Table 1 lists the format of a message. The first three flits of the message contain the X, Y, and Z addresses. Each node along the path compares the address in the head flit of the message with the node's index in the current dimension. If the two indices match, the node strips the head flit off the message and routes the rest to the next dimension. The MDP's network output node formats the address flits of the message. It also precomputes the direction (positive or negative) the message must travel along each dimension, setting additional bits in the address flits. This reduces the latency and complexity of the router nodes.

The network uses blocking flow control to resolve contention for a physical channel (see Figure 13). When a message arrives at a router path already in use by a message of the same priority, it is blocked. The blocked message compresses into routers along its path, occupying one node per word (two flits) of the message. When the blockage clears, the message uncompresses and proceeds to its destination, at a rate of one hop per cycle.

Two priorities of messages share the physical wires, but use completely separate buffers and routing logic. This allows priority 1 messages to proceed through blockages at priority 0. Without this ability, the system could not redistribute data that has caused hot spots in the network.

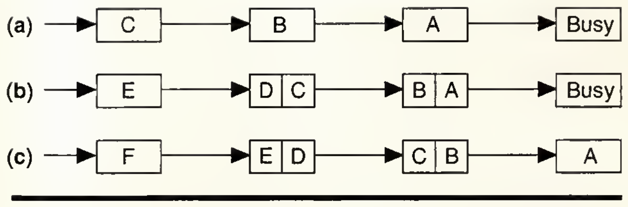


Figure 13. The J-Machine network performs blocking flow control with two stages of queueing per node. Message arrives at busy channel (a). Message becomes compressed by queueing (b). Channel is available; message continues advancing (c).

Table 1. A typical message in the J-Machine.

Flit	Contents	Remark
1	5:+	X address
2	1:–	Y address
3	4:+	Z address
4	MSG: 00	Method to call
5	00440	
6	INT: 00	Argument to method
7	0023	
8	INT: 00	Reply address
9	<1:5:2> T	

The first three flits contain the destination address. The final flit in the message is marked as the tail.

MDP implementation

Figure 12 shows the major subsystems in the MDP. The chip includes a conventional microprocessor with prefetch, control, register file and ALU (RALU), and memory blocks. The communication system comprises the routers and network input and output interfaces. The address arithmetic unit (AAU) provides addressing functions. The MDP also includes a DRAM interface, control block, and diagnostic interface.

Communication subsystem. The communication subsystem contains the network output, the network input, and the routers. The network output block buffers messages from the registers or memory and injects them into the network. A FIFO buffer matches the speed of message transmission to the network. On each SEND instruction, the MDP transfers one or two words to its FIFO. When the message is complete, or the eight-word buffer is full, the buffer launches the message into the network. In cases where the MDP cannot send message words as fast as the network can transmit them, the FIFO prevents bubbles (absence of words) from entering the network pipeline and degrading performance.

The network input module transfers messages from the network to the MDP's memory. Data from the network arrive in 18-bit flits, which are composed into a four-word queue row buffer. When the QRB fills, it writes its contents to the on-chip memory in one cycle. Writing memory a row (4 × 36 bits) at a time reduces the number of memory cycles consumed by the network, leaving more memory bandwidth for the CPU.

The routers form the switches in a J-Machine network and

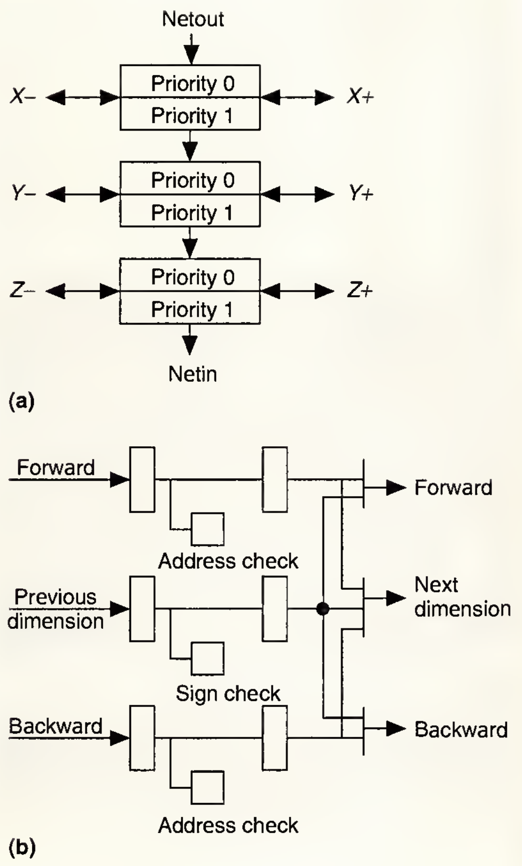


Figure 14. Block diagram of the routers. The two priorities per dimension are completely separate except where they share physical channels (a). Each priority contains forward, reverse, and previous to next dimension datapaths (b).

deliver messages to their destinations. As shown in Figure 14a, the MDP contains three independent routers, one for each bidirectional dimension of the network. Each router contains two separate virtual networks with different priorities that share the same physical channels. The priority 1 network can preempt the wires even if the priority 0 network is congested or jammed.

Each of the 18 router paths contains buffers, comparators, and output arbitration (Figure 14b). On each data path, a comparator compares the lead flit, which contains the destination's address in this dimension, to the node coordinate. If the head flit does not match, the message continues in the current direction. Otherwise the message is routed to the next dimension. Messages entering the dimension compete with messages continuing in the dimension at a two-to-one switch. Once a message is granted this switch, any other input is locked out for the duration of the message. Once the head flit of the message has set up the route, subsequent flits follow directly behind it.

Address arithmetic unit. The AAU, the largest logic block in the MDP, performs all functions associated with memory addressing. To support naming and relocation, the AAU contains the address and ID registers. It protects memory accesses and implements the translation instructions. Each memory reference is offset by the selected address register's base field and checked against its length field. An attempt to access through an invalid address register (which may occur when an object relocates) or to access beyond the end of an object raises an exception. A translation base/mask register defines an area of memory to be a two-way, set-associative translation buffer used by the XLATE, PROBE, and ENTER instructions. The AAU hashes the keys used to access this table using an exclusive-Or network to improve hit rate in the translation buffer.

The AAU maintains two queues to buffer incoming messages and schedule the associated tasks. Associated with each queue are a queue base/mask (QBM) and a queue head/length (QHL) register. (See Figure 15.) The QBM registers define the position and length in main memory of the message queues. Queues are circular, so messages at the end of the queue wrap around to the beginning. The QHL registers point to the beginning of the first message in the queue and its length field encompasses exactly all of the messages currently in the queue. When the MDP dispatches a task to handle a message, it loads the A3 register with a segment descriptor for the message. The processor dispatches a task as soon as the first four words of a message are written. If the task attempts to read a word of the message which has not yet arrived, a special Early fault occurs.

Layout. Figure 16 shows a floor plan of the chip with a die photograph for comparison. Table 2 breaks down the area usage.

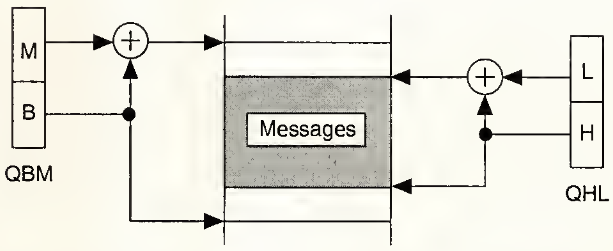
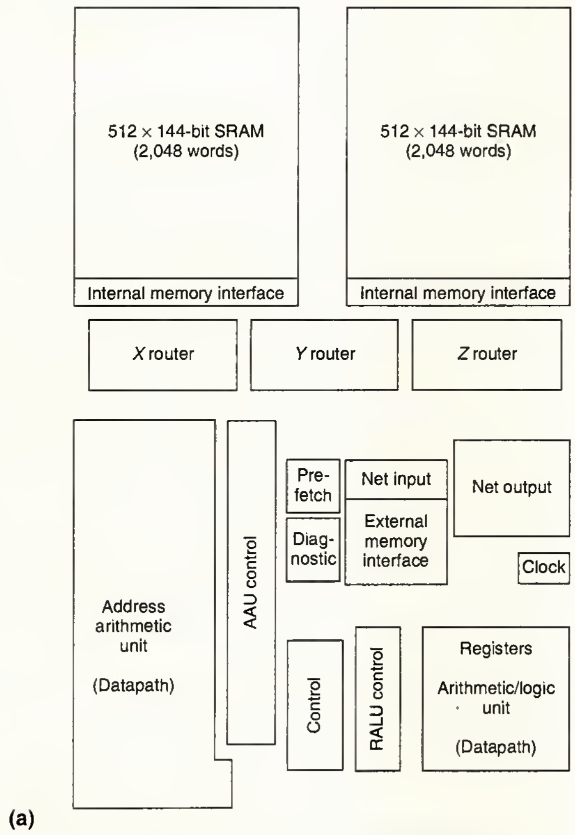


Figure 15. The AAU maintains the queue base/mask (QBM) registers, which specify the location of the message queues in main memory, and the queue head/length (QHL) registers, which specify the beginning and end of the messages received in each queue. Figure shows only one queue.


512 × 144-bit SRAM (2,048 words)
Internal memory interface
512 × 144-bit SRAM (2,048 words)
Internal memory interface
X router
Y router
Z router
Address arithmetic unit (Datapath)
AAU control
Pre-fetch
Diag-nostic
Net input
External memory interface
Net output
Clock
Control
RALU control
Registers
Arithmetic/logic unit
(Datapath)
(a)

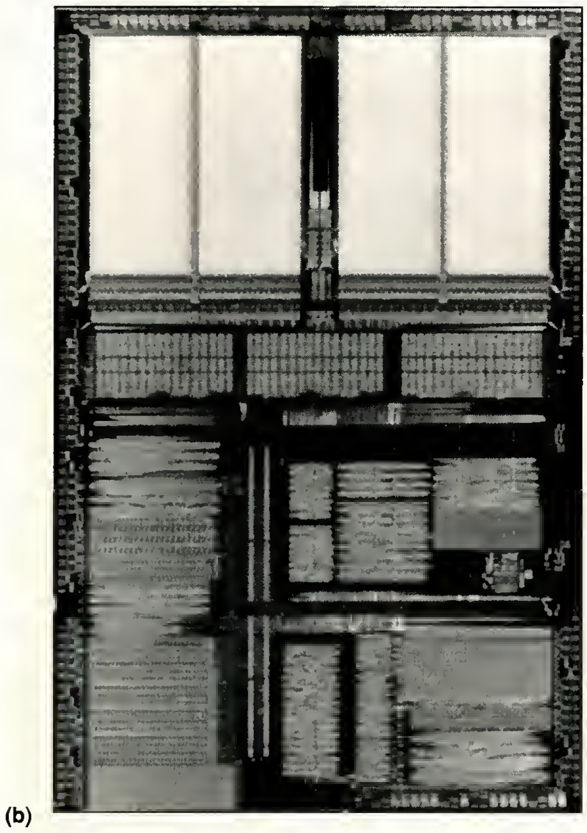


Figure 16. MDP chip floor plan (a) and die photograph (b).

Methodology. We implemented the MDP using Intel standard cells except for the on-chip RAM, clock generator, and pads. Using standard cells sacrificed a factor of three to four in area and two to three in performance over what would be possible with full-custom design. The advantage was a significant increase in productivity which was essential to completing the chip successfully with our small design team.

The 700 or so sheets of schematics drafted at MIT used 35,000 standard cells containing 210,000 transistors. (The remaining 890,000 devices are contained in the full custom portions of the chip, mostly in the RAM.) We sent these schematics to Intel for layout. Designers laid out many of the data paths by hand to exploit the regularity of the design. Automatic place and route CAD tools laid out the less regular collections of logic.

We began architecture studies leading to the MDP in October 1986. Work on the RTL model of the microarchitecture began in June 1988, and schematic entry at MIT started that November. The task of translating schematics into layout commenced in June 1989, and we finished the layout in December 1990. We received first silicon in June 1991 and were running programs on it within a few hours.

Table 2. Chip area breakdown.

Module	Dimensions (mm)	Area (mm^2)	Transistors ($\times 10^3$)
AAU	3.7 × 7.0	25.9	75.0
RALU	3.7 × 2.9	10.7	39.0
Diagnostic	0.9 × 1.1	1.0	3.7
Prefetch	0.9 × 1.1	1.0	3.2
Control	1.1 × 2.6	2.9	8.7
Internal memory interface	7.8 × 0.5	3.9	13.0
External memory interface	1.6 × 1.8	2.9	9.0
Net input	1.8 × 0.7	1.3	4.4
Net output	2.1 × 1.8	3.8	18.0
Routers	8.4 × 1.3	10.9	29.0
RAM	8.8 × 4.9	43.1	880.0
Clock	0.7 × 0.8	0.6	0.1
Pads	50.5 × 0.2	8.4	2.6
Full chip	10.2 × 15.0	153.0*	1,087.0

* Includes wiring between modules.

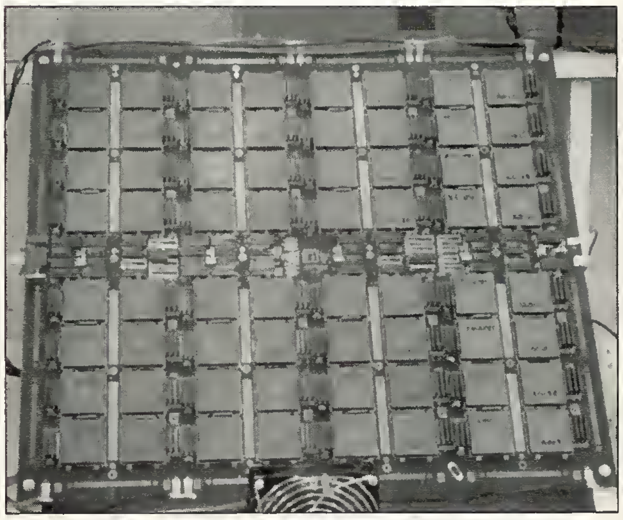


Figure 17. Photograph of 64-node J-Machine system.

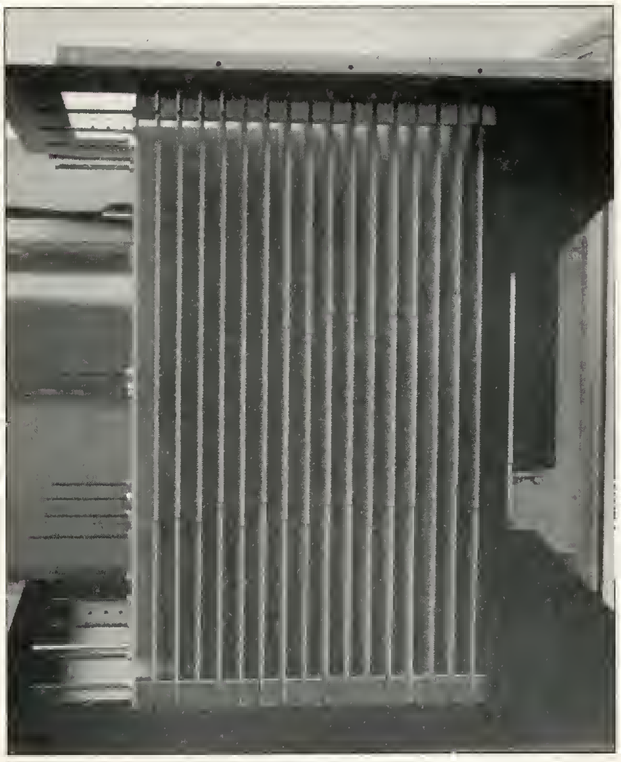


Figure 18. A 1,024-node J-Machine chassis.

Although we thoroughly simulated the logic design, we have uncovered 12 bugs while running our validation tests and applications on the hardware. Some of these bugs have simple software work-arounds, but for performance reasons we sent a second revision of the layout with modified control logic and some metal fixes for fabrication in January of this year. We plan to use several thousand of these chips to build research multicomputers at MIT.

System design. Figure 17 shows a photograph of a 64-node J-Machine processor board measuring 20.5 in. × 24 in. Each node consists of an MDP chip (in a 168-pin grid array package) and three 4-Mbit DRAMs. Each pair of nodes shares a set of elastomeric connectors to communicate with the corresponding nodes on the boards above or below the board in a stack. A total of 32 elastomeric connectors held in four connector holders provide 2,240 electrical connections between adjacent boards. Of these connections, 960 are used for signalling and the remaining are ground returns. No power is supplied through the elastomers. Bus bars supply power and ground directly to each board. The center area of the board contains the final stage of the clock distribution network, along with diagnostic fan-out, multiplexing logic, and temperature and airflow monitors.

Figure 18 shows a photograph of our chassis for a 1,024-node system. The chassis contain a stack of 16 processor boards, power supplies, and distribution bus bars. Twenty tie rods bind the boards and compress the elastomer connectors. A 4,096-node system can be built by combining four chassis. Each stack connects to its neighboring stacks by 128 (16 × 8) short, 60-pin, ribbon cables—one for each pair of nodes on the periphery. Each vertical pair of stacks shares a 3,000 cu ft/min. blower for cooling.

In addition to the processor board and chassis, we have also designed a diagnostic interface board and are designing a SCSI disk interface, a distributed graphics frame buffer, and an S-bus interface. Noakes and Dally[22] offer more details of the J-Machine system design.

Software

We intended the J-Machine as a platform for software experiments in fine-grain, parallel programming. To this end, we have implemented and are studying software systems for different fine-grain programming models. Fine-grain programs typically execute from 10 to 100 instructions between communication and synchronization actions. Reducing the grain size of a program increases both the potential speedup due to parallel execution and the potential overhead associated with parallelism. Special hardware mechanisms to reduce the overhead due to communication, process switching, synchronization, and multithreading are therefore central to the design of the MDP. Software issues such as load balancing, scheduling, and locality remain open questions and are the focus of current research efforts.

```
(defmethod Size-Of-Tree Pair ( )
   (+ (Size-Of-Tree Left)
      (Size-Of-Tree Right)))
(defmethod Size-Of-Tree Object ( )
   1)
(defmethod Size-Of-Tree Null ( )
   0)
```

Figure 19. Concurrent Smalltalk source to compute Size-Of-Tree. Method definitions specify the class to which they apply. The class Pair contains two elements, Left and Right, each of which may hold an Object or another Pair.

A parallel processor creates programming challenges. It is difficult to extract the fine-grain parallelism needed from stock programs written in C or Fortran. Instead of concentrating on extracting parallelism from existing programs (an active and interesting area for many parallel programming researchers) or on adapting sequential languages for the parallel domain, we focus on languages where the expression of fine-grain parallelism is much cleaner. To date, we have implemented two languages on the J-Machine: the actor language Concurrent Smalltalk and the dataflow language Id.

Concurrent Smalltalk. CST[23] is a parallel, object-oriented, programming language (based on the Actor model[5]) with asynchronous message send and distributed objects. Its syntax is similar to that of Lisp or Scheme. It performs method or function invocation by sending a message to the first argument of the method. The message contains the method selector and the rest of the arguments.

Functions and methods in the language are compiled into MDP assembly code by an optimizing compiler, called Optimist, and assisted at runtime by a small kernel called Cosmos. Cosmos provides a global virtual name space, object-based memory management, support for distributed objects, and low-overhead context switching. Its memory management system provides fast, transparent access to storage distributed across the machine. Cosmos efficiently supports fine-grain concurrent computation in which tasks are very short (40 user instructions) and data objects are very small (eight words). The CST compiler and the Cosmos runtime system also provide floating-point arithmetic, simple arrays, and garbage collection for CST programs. Cosmos manages contexts, futures, and objects, and therefore plays an important role in providing services that exploit the communication, synchronization, and naming mechanisms of the J-Machine.

Figure 19 shows a small sample program defining the Size-Of-Tree method for three object types: a Pair, the Null object, and a generic object. When called on a Lisp-style tree, these methods return the number of generic objects stored in the tree. For example, when called on the tree '((1 2 3)(4 5 6)(7 8 9)), Size-Of-Tree returns the value 9. Note that since Pair and Null are subclasses of Object, their more specific methods are selected when Size-Of-Tree is invoked on their types.

When Optimist, the CST compiler, compiles this example program, it defines a selector object and three function objects. The selector object (shown in Figure 20) lists the type and function correspondence. When a method applies to an object, Cosmos examines the object type and locates the appropriate function in the selector object. The MDP then invokes this function on the object. (In cases where the compiler can infer the type of the object or when the type of objects is explicitly declared, the compiler optimizes a method invocation directly to the correct function invocation.) The compiler marks the selector object as copyable, and Cosmos maintains it like any other object.

Figure 21 shows the compiled code for the function for the class Pair. When a method applies to a particular object, Cosmos examines the object class and the selector object, and chooses the correct function to invoke.

The function first does an XLATE operation to get the address of the Pair and uses that address to get the object ID for Left. It then calls Cosmos to find the node where Left exists. The function sends a message to Left that recursively applies the Size-Of-Tree method. It marks the slot that will hold the return value with a Cfut tag. Next, it applies Size-Of-Tree to Right without waiting for the result of the first remote procedure to return. However, when the function attempts to add the two return values, the results will probably not have returned yet. In this case, the ADD instruction will fault trying to add Cfuts, and the MDP will suspend the process, saving its registers into the context.

```
MODULE   OBJ:Selector.Size_Of_Tree

DC       Copyable | class_Selector     ; Identify properties of
                                       ;   Size_Of_Tree selector
DC       OBJ:Selector.Size_Of_Tree     ; Store own ID inside selector

DC       3                             ; Number of functions
DC       CLASS:Object                  ; Class identifier for Object
DC       {function.Size_Of_Tree}       ; Function for class Object
DC       CLASS:Null                    ; Class identifier for Null
DC       {function.Size_Of_Tree_1}     ; Function for class Null
DC       CLASS:Pair                    ; Class identifier for Pair
DC       {function.Size_Of_Tree_2}     ; Function for class Pair
```

Figure 20. Selector object generated by the example program.

```
MODULE      OBJ:function.Size_Of_Tree_2

    DC          Copyable|class_Function          ; Identify properties of Size_Of_Tree function
    DC          {OBJ:function.Size_Of_Tree_2}    ; Store own ID inside function

    ;; Incoming:   A1 points to the context
    ;;             A3 points to the message

START:
    MOVE        [2,A3],R3                        ; Get the Pair's object ID
    XLATE       R3,A2                            ; Find the Pair's local address
    MOVE        [2,A2],R0                        ; Get the object ID of Left
    CALL        objectNode,R1                    ; Find Left's node -> R1
    DC          MSG:Apply_Selector               ; Send a message to Left to apply
    SEND2       R1,R0                            ;   the method specified by the
    DC          {OBJ:Selector.Size_Of_Tree}      ;   selector for Size_Of_Tree.
    SEND        R0                               ; Includes our context ID
    SEND        [2,A2]                           ;    and a continuation.
    MOVE        5,R0
    SEND2E      [1,A1],R0
    WTAG        R0,CFUT,R0                       ; Make a future for Left's
    MOVE        R0,[5,A1]                        ;   result.
    MOVE        [3,A2],R0                        ; Do the same for Right as for
    CALL        objectNode,R1                    ;   Left.
    DC          MSG:Apply_Selector
    SEND2       R1,R0
    DC          {OBJ:Selector.Size_Of_Tree}
    SEND        R0
    SEND        [3,A2]
    MOVE        6,R0
    SEND2E      [1,A1],R0
    WTAG        R0,CFUT,R0
    MOVE        R0,[6,A1]
    MOVE        [6,A1],R2                        ; Do the sum.
    ADD         R2,[5,A1],R1
    MOVE        [3,A3],R3                        ; Get the continuation for
    BNIL        R3,^L001                         ;   this context.  Reply if
    DC          MSG:Reply                        ;   non-nil.
    SEND2       R3,R0
    SEND        R3
    SEND2E      [4,A3],R1
L001:
    SUSPEND
    END
```

Figure 21. Compiled code for the Size-Of-Tree function for objects of class Pair.

Assuming this happens, when the MDP receives replies from the methods after writing the value into the future slot, Cosmos checks to see if the process was waiting for that particular future. If so, it reactivates the context. The reactivated function would then sum the two results and forward them to the continuation specified in the original method invocation.

Let us consider some interesting points:

- If the object of the function is not present or if the translation cache does not have an entry for the object, the

XLATE instruction will fault. Cosmos will find the object and move or copy it to the local node.

- Cosmos maintains functions and selectors like any other immutable object. If they are not present, Cosmos will copy them to the node, a process analogous to a distributed instruction cache.
- If the function were preempted and the object moved or migrated away, Cosmos would invalidate the address registers. Accesses to the object would cause a fault that would attempt to retranslate or reobtain the object.
- The A1 register points to the current context. The context contains storage to hold working variables or, if the context faults, to hold spilled register values. In the example, the futures are constructed in the context, and thus are named *context-future* (Cfut).

This example illustrates some important research questions related to the efficiency of this model of computation.

- When is it better to spawn processes nonlocally rather than locally? This is probably a strong function of the amount of associated overhead. The MDP architecture attempts to reduce this overhead, but algorithms for making this trade-off at compile and runtime still need to be developed and evaluated.
- How should we place objects in the machine, and how should they migrate in order to reduce the overhead of communication?
- In some cases, the amount of parallelism grows much larger than the machine can handle. We need to study how we can effectively and automatically throttle the parallelism created by the machine when it becomes saturated.

Horwat discusses these issues, and others related to the efficiency of programming fine-grain, parallel processors in more detail.[23]

Dataflow implementation. Id is a functional programming language originally designed for dataflow architectures.[24] The Id compiler converts an Id program into a dataflow graph, in which nodes represent operators and arcs represent dependencies. Originally, researchers executed these dataflow graphs directly on specialized dataflow machines. More recently, they have begun compiling dataflow graphs to run on general-purpose parallel machines.[25] Dataflow programs suit large parallel computers, because the abundance of fine-grain tasks—each of which can be as small as a single dataflow operator—makes it easy to mask communication latency with task switches. Conversely, the J-Machine's fine-grain mechanisms make it an excellent target for dataflow programs.

We experimented with several methods of executing dataflow programs on the J-Machine.[26] The simplest of the systems translates each node of the dataflow graph into a sequence of MDP instructions. A dataflow node with two inputs takes 20 MDP instructions to simulate. To do so, it stores the first data value, matches it with the second value when it arrives, performs the dataflow operation, and sends the resulting value to two destinations. This process uses the Cfut tag and fault handler.

A more efficient approach increases the granularity of each task to reduce scheduling overhead. We are building a system on top of the Berkeley TAM project[25] that addresses the inefficiencies of our earlier systems.

WE BUILT THE MDP TO DEMONSTRATE THE UTILITY of general-purpose communication, synchronization, and naming mechanisms in a multicomputer building block. Its mechanisms efficiently support dataflow[26] and object-oriented programming[23] models using a global name space. The use of a few simple mechanisms provides orders of magnitude lower communication and synchronization overhead than is possible with multicomputers built from off-the-shelf microprocessors. Its communication and synchronization performance competes with processing nodes specialized to a single model of computation, such as iWarp[15] (systolic) or the transputer[13] (communicating sequential processes).

Computers built from fine-grain processing nodes, such as the MDP, consisting of a small but powerful processor and a small memory, are more cost-effective than those built from fewer coarse-grain nodes. Fine-grain nodes devote a larger fraction of their silicon area to processing and have higher arithmetic, memory, and communication bandwidth per unit cost. Large-scale parallel machines built from fine-grain processors have a larger total amount of memory within a given latency of a processor. An efficient network design provides global memory latency and bandwidth competitive with coarse-grain machines.

The MDP is a component for building scalable computer systems. It is useful in configurations ranging from one node to 65,536 nodes. A 128-node Jellybean Machine is currently operational and resources are in place to build several more machines, including a 1,024-node system at MIT and machines at a number of other research institutions.

The MDP project demonstrated the feasibility of building experimental computer systems with limited resources. By concentrating on the novel mechanisms of the MDP and keeping the design simple and modest in other respects, we completed the design of the chip, its system-level hardware, and several programming systems with a handful (less than eight) of graduate students and engineers in two and a half years.

With the MDP we have begun exploring mechanisms for parallel computers. Much work remains to be done to tune the MDP's mechanisms and compare them to alternatives. The demands of parallel software that drive these mechanisms are very different from the demands placed on sequential computers. We find the design of mechanisms for parallel computers particularly challenging because no well-established parallel benchmarks exist. Additionally, most parallel programs are very biased by the mechanisms (or lack thereof) of the machines for which they were initially written.

Our software studies have suggested improvements that could be made to the MDP. More registers and better mapping mechanisms would be useful. MDP's conservative implementation leaves opportunities for streamlining, by decreasing the cycle time and number of clocks per instruction. A commercial, custom VLSI product based on the architectural mechanisms in the MDP is very plausible.

As technology scales, we can put many powerful processing units on one chip. An interesting direction for further research is the extension of the MDP mechanisms to control intranode as well as internode concurrency. The MIT M-Machine project, now in its early phase, takes this approach. It employs a processor-coupling mechanism to allow local processors to interact with single-cycle latency.

Acknowledgments

The MDP was built as part of a collaborative project between MIT and Intel Corporation. The Defense Advanced Research Projects Agency partially funded the project, under contracts N00014-88K-0738 and N00014-91-J-1698. Funding also came from a National Science Foundation Presidential Young Investigator Award, grant MIP-8657531, with matching funds from General Electric Corporation, IBM Corporation, and AT&T Corporation. Intel Corporation provided technical support and personnel.

Andrew Chien, Waldemar Horwat, and D. Scott Wills helped develop the MDP architecture. Scott Furman, Shaun Kaneshiro, Ellen Spertus, Brian Totty, and Deborah Wallach were instrumental in developing the J-Machine system software. Salim Ahmed, Paul Carrick, Steve Lear, Ted Nguyen, and Mark Vestrich of Intel Corporation provided engineering and management support. Discussions with Chuck Seitz, Steve Ward, Anant Agarwal, and Tom Knight were helpful during the early stages of the J-Machine project. We thank Lisa Sardegna for her help preparing this manuscript.

References

1. W.J. Dally, D.S. Wills, and R. Lethin, "Mechanisms for Parallel Computing," *Proc. NATO Advanced Study Institute on Parallel Computing on Distributed Memory Multiprocessors*, Springer-Verlag, New York, 1991.
2. A. Gottlieb et al., "The NYU Ultracomputer – Designing a MIMD Shared Memory Parallel Computer," *IEEE Trans. Computers*, Vol. C-32, No. 2, Feb. 1983, pp. 175-189.
3. W.D. Hillis and G.L. Steele, "Data Parallel Algorithms," *Comm. of the ACM*, Vol. 29, No. 12, 1986, pp. 1170-1183.
4. A.H. Veen, "Dataflow Machine Architecture," *ACM Computing Surveys*, Vol. 18, No. 4, Dec. 1986, pp. 365-396.
5. G. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems," Tech. Report 844, Artificial Intelligence Laboratory, Massachusetts Inst. of Technology, Cambridge, Mass., 1985.
6. W.C. Athas and C.L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *Computer*, Vol. 21, No. 8, Aug. 1988, pp. 9-24.
7. W.J. Dally, "Performance Analysis of *k*-ary *n*-cube Interconnection Networks," *IEEE Trans. Computers*, Vol. 39, No. 6, June 1990, pp. 775-785.
8. R. Arlauskas, "iPSC/2 System: A Second-Generation Hypercube," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, Assn. for Computing Machinery, New York, 1988, pp. 33-36.
9. J.F. Palmer, "The Ncube Family of Parallel Supercomputers," *Proc. IEEE Int'l Conf. Computer Design*, IEEE CS Press, Los Alamitos, Calif., 1986, p. 107.
10. *Series 2010 Product Description*, Ametek Computer Research Division, Monrovia, Calif., 1987.
11. "Butterfly Parallel Processor Overview," BBN Report 6148, Bolt, Beranek and Newman Advanced Computers, Inc., Cambridge, Mass., Mar. 1986.
12. W.C. Brantley, K.P. McAuliffe, and J. Weiss, "RP3 Processor-Memory Element," *IEEE Trans. Computers*, Vol. C-34, No. 10, Sept. 1985, pp. 782-789.
13. I. Barron et al., "Transputer Does Five or More MIPS Even When Not Used in Parallel," *Electronics*, Nov. 1983, pp. 109-115.
14. C. Lutz et al., "Design of the Mosaic Element," *Proc. MIT Conf. Advanced Research in VLSI*, Artech Books, Dedham, Mass., 1984, pp. 1-10.
15. S. Borkar et al., "iWarp: An Integrated Solution to High-Speed Parallel Computing," *Proc. Supercomputing Conf.*, IEEE CS Press, Nov. 1988, pp. 330-338.
16. W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *Distributed Computing*, Vol. 1, 1986, pp. 187-196.
17. W.J. Dally and P. Song. "Design of a Self-Timed VLSI Multicomputer Communication Controller," *Proc. Int'l Conf. Computer Design*, pp. 230-234. IEEE CS Press, Oct. 1987.
18. P.C. Yew, N.F. Tzeng, and D.H. Lawrie, "Distributing Hot-Spot Addressing in Large-Scale Multiprocessors," *IEEE Trans.*

Computers, Vol. C-36, No. 4, Apr. 1987, pp. 388-395.

19. W.J. Dally et al., "Architecture of a Message-Driven Processor," *Proc. 14th Int'l Symp. Computer Architecture*, IEEE CS Press, June 1987, pp. 189-205.
20. P.R. Nuth, "Router Protocol," internal memo 23, MIT Concurrent VLSI Architecture Group, 1990.
21. W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, Vol. C-36, No. 5, May 1987, pp. 547-553.
22. M. Noakes and W.J. Dally, "System Design of the J-Machine," *Proc. Sixth MIT Conf. Advanced Research in VLSI*, W.J. Dally, ed., MIT Press, Cambridge, Mass, 1990, pp. 179-194.
23. W. Horwat, "Concurrent Smalltalk on the Message-Driven Processor," master's thesis, MIT, May 1989.
24. R.S. Nikhil, *ID Version 88.1 Reference Manual*, Tech. Report 284, Computation Structure Group, MIT, 1988.
25. D.E. Culler et al., "Fine-Grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract Machine," *Proc. Fourth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 1991, pp. 164-175.
26. E. Spertus and W.J. Dally, "Experiments with Dataflow on a General-Purpose Parallel Computer," *Proc. Int'l Conf. Parallel Processing*, Aug. 1991, pp. II-231–II-235.

William J. Dally is an associate professor of computer science at the Massachusetts Institute of Technology, where he directs the group building the J-Machine. His research interests include concurrent computing, computer architecture, computer-aided design, and VLSI design.

Dally received a BS degree from Virginia Polytechnic Institute and an MS from Stanford University, both in electrical engineering. He earned his PhD in computer science from the California Institute of Technology.

J.A. Stuart Fiske is a PhD student at MIT. His main research interests include computer architecture, parallel processing, and VLSI design. He received the BE degree in electrical engineering from McGill University, and the MS in electrical engineering and computer science from MIT. Fiske is a member of the IEEE.

John S. Keen is a PhD student at the MIT AI Lab. He has worked at IBM Canada, Bell Canada, and Intel. His primary research interests include computer architecture, parallel processing, CAD tools, and concurrent database systems.

Keen graduated from the University of Waterloo with a BASc degree and an MASc degree, both in electrical engineering. He received a National Science and Engineering Research Council of Canada 1967 Scholarship and is a member of the IEEE and Sigma Xi.

Richard A. Lethin is also pursuing his PhD at MIT's AI Lab. Previously, he designed floating-point processor boards for the Trace-200 and Trace-300 VLIW (very long instruction word) mini-supercomputers at Multiflow Computer. His research interests include computer architecture and artificial intelligence.

Lethin received a BS degree in electrical engineering from Yale College and an MS in electrical engineering and computer science from MIT. He is a Hertz Foundation fellow.

Michael D. Noakes is a member of the research staff of MIT's AI Lab, where he researches parallel computer architecture. Previously he helped develop the Concert multiprocessor at Harris Corporation. His research interests include computer architecture and parallel processing. Noakes earned a BS degree in electrical engineering from MIT.

Peter R. Nuth is a PhD student in the MIT AI Lab. He helped design MIT's Concert and J-Machine parallel computers. His primary research interests include computer architecture, communication, and parallel processing.

Nuth earned BS, MS, and Engineer's degrees in electrical engineering and computer science from MIT. He belongs to the IEEE, Sigma Xi, and Eta Kappa Nu.

Roy E. Davison, a 25-year veteran of the IC industry, heads Davison Design and Development in Ben Lomond, California. He has designed chips for several manufacturers, including Texas Instruments, Advanced Micro Devices, National Semiconductor, Intel, and Mas Par. His primary research interest is the design of microprocessors and peripherals. He attended Sam Houston State University.

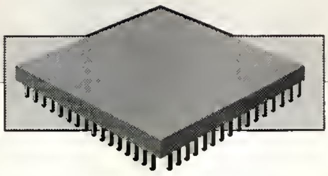
Gregory A. Fyler is a project manager in Intel's Architecture Development Lab in Santa Clara, California. He has managed chip design for several microcontroller and microcomputer products, most recently the MDP project. Prior to joining Intel 15 years ago, he developed calculator chips for Fairchild Camera and Instrument.

Fyler earned a BS degree in electrical engineering and computer science from the University of California, Berkeley. He is a member of the IEEE.

Direct questions regarding this article to William J. Dally, MIT Artificial Intelligence Lab, 545 Technology Square, NE43-620, Cambridge, MA 02139; or via e-mail at billd@ai.mit.edu

Reader Interest Survey
Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156	Medium 157	High 158



Organization of the Motorola 88110 Superscalar RISC Microprocessor

Motorola's second-generation RISC microprocessor employs advanced techniques for exploiting instruction-level parallelism, including superscalar instruction issue, out-of-order instruction completion, speculative execution, dynamic instruction rescheduling, and two parallel, high-bandwidth, on-chip caches. Designed to serve as the central processor in low-cost personal computers and workstations, the 88110 supports demanding graphics and digital signal processing applications.

Keith Diefendorff

Michael Allen

Motorola

Motorola designers conceived of the 88000 RISC (reduced instruction-set computer) architecture to simplify construction of microprocessors capable of exploiting high degrees of instruction-level parallelism without sacrificing clock speed. The designers held architectural complexity to a minimum to eliminate pipeline bottlenecks and remove limitations on concurrent instruction execution.

The 88100/200 is the first implementation of the 88000 architecture. It is a three-chip set, requiring one CPU (88100) chip and two (or more) cache (88200) chips. The CPU's simple scalar design uses multiple concurrent execution units with out-of-order instruction completion to approach a throughput of one instruction per clock cycle.

The second-generation, single-chip 88110 RISC microprocessor employs superscalar instruction issue and out-of-order instruction execution techniques to achieve a throughput greater than one instruction per clock cycle.

Overview

In designing the 88110, we aimed at a general-purpose microprocessor, suitable primarily for use as the central processor in low-cost personal computers and workstation systems. Thus, our design objective was good performance at a given cost, rather than ultimate performance at any cost. We recognized that the personal computer environment is moving toward highly interactive software, user-oriented interfaces, voice and image processing, and advanced graphics and video, all of which would place extremely high demands on integer, floating-point, and graphics processing capabilities. At the same time, we realized the 88110 would have to meet these performance demands while operating with the inexpensive DRAM systems typically found in low-cost personal computers.

To achieve the performance goals set for the 88110, we needed to obtain more parallelism than was achieved in earlier microprocessors. To this end, we decided to use a superscalar microarchitecture to exploit additional instruction-level parallelism. Superscalar machines are distinguished by their ability to dispatch multiple instructions each clock cycle from a conventional linear instruction stream. This approach has shown good speedup on general-purpose applications and was a good match to available CMOS technology. (We believe Agerwala and Cocke coined the superscalar term.[1])

We selected the superscalar approach over

0272-1732/92/0400-0040$03.00 © 1992 IEEE

other fine-grain parallelism approaches, such as the vector machine and the VLIW (very long instruction word) approach, because it appeared to be more effective for our intended application. With a limited transistor budget, spending transistors on vector hardware would have meant sacrificing scalar performance and would have yielded a machine that suffered from the Amdahl's Law phenomenon[2] on general-purpose applications.

The VLIW approach[3] would have introduced severe software compatibility restrictions, by exposing hardware parallelism to the object code program and thereby limiting future implementation flexibility. Also, the VLIW speedup, while substantial on code with abundant parallelism (such as scientific applications), is less significant on general-purpose applications. This limited speedup is due in part to the code expansion: The inefficient use of opcode bits to control unused execution units and the aggressive loop unrolling required to schedule the available execution unit parallelism effectively.[4]

The superscalar approach appeared to be a better match to CMOS technology than a superpipelined approach. Superscalar designs rely primarily on spatial parallelism—multiple operations running concurrently on separate hardware—achieved by duplicating hardware resources such as execution units and register file ports. Superpipelined designs, on the other hand, emphasize temporal parallelism—overlapping multiple operations on a common piece of hardware—achieved through more deeply pipelined execution units with faster clock cycles. As a result, superscalar machines generally require more transistors, whereas superpipelined designs require faster transistors and more careful circuit design to minimize the effects of clock skew. Some literature indicates that superscalar and superpipelined machines of the same degree would perform roughly the same.[5] We felt that CMOS technology generally favors replicating circuitry over increasing clock cycle rates, since CMOS circuit density historically has increased at a much faster rate than circuit speed.

The 88110 microarchitecture, illustrated in Figure 1, employs a symmetrical superscalar instruction dispatch unit, which dispatches two instructions each clock cycle into an array of 10 concurrent execution units. The design implements fully interlocked pipelines and a precise exception model, but it allows out-of-order instruction completion, some out-of-order instruction issue, and branch prediction with speculative execution past branches.

We optimized each execution unit for low latency: The branch unit uses a branch target instruction cache to reduce branch latency. The integer and graphics units are one-cycle units; the floating-point adder and multiplier are three-cycle, fully pipelined, IEEE extended-precision units. The load/store unit provides fast access to the cache on a hit. But it is also highly buffered to increase tolerance to long memory latency on a miss and to allow dynamic reordering of loads and stores for runtime overlapping of tight loops.

The on-chip caches are organized in a Harvard arrangement, giving the processor simultaneous access to instructions and data. Each 8-Kbyte, two-way, set-associative cache provides 64 bits each clock cycle to its respective unit. The write-back data cache is nonblocking for some types of accesses, and it follows a four-state MESI (modified, exclusive, shared, invalid) protocol[6] for coherence with other caches in multiprocessor systems. It also supports selective cache bypass and software prefetching from user mode. Two independent, 40-entry, fully associative address translation look-aside buffers (TLBs) support a demand-paged virtual

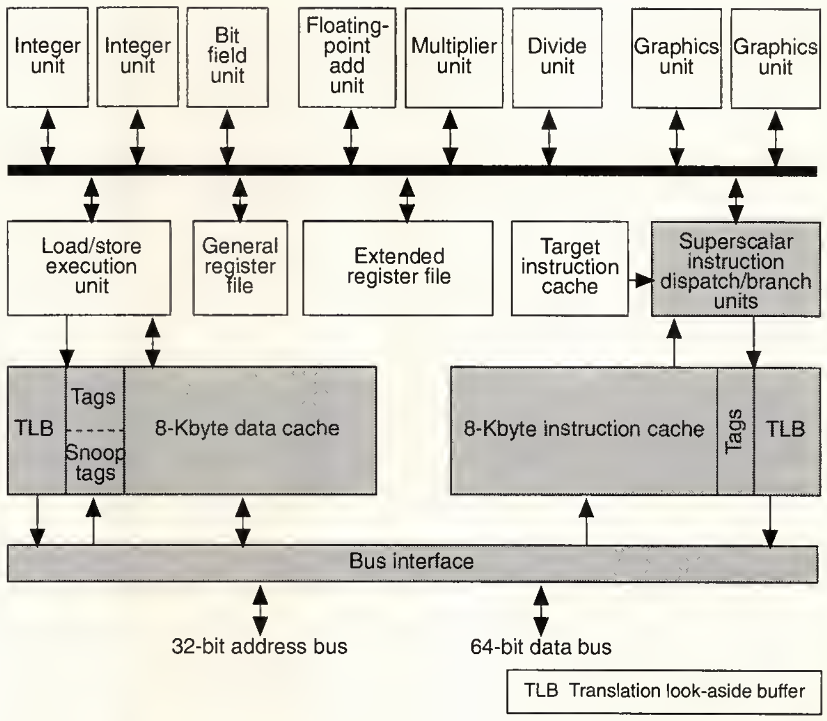


Figure 1. Block diagram of the 88110.

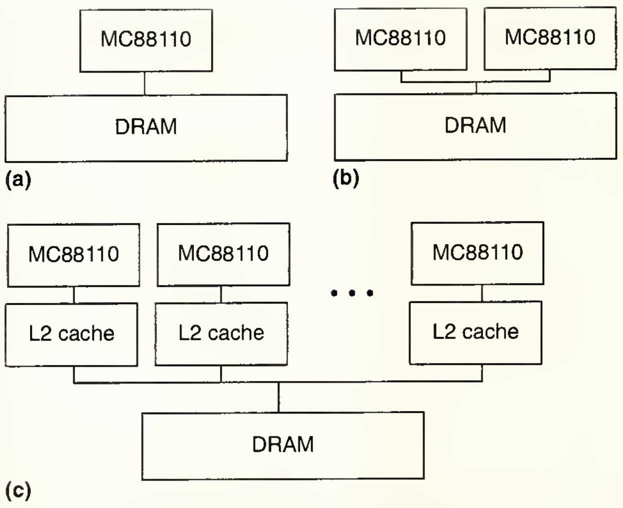


Figure 2. Target system configurations: single (a), dual (b), and multiprocessors (c).

memory environment. A common, 64-bit, external bus services cache misses. The demultiplexed, pipelined bus supports burst mode, split transactions, and bus snooping.

We designed the 88110 especially for the three basic system configurations shown in Figure 2:

- single processors tightly coupled to low-cost DRAMs;
- dual-processor systems, also coupled to inexpensive DRAMs, either in a symmetrical multiprocessing arrangement or with one of the 88110s dedicated to a particular function such as graphics or digital signal processing (DSP); and
- medium-scale shared-memory multiprocessor systems, with each processor using local secondary static RAM (SRAM) cache, which we call L2.

Instruction set architecture

To improve performance, we extended the instruction set architecture of the 88110 beyond that of the 88100 microprocessor. We enhanced a number of the integer and floating-point instruction sets and added a new set of capabilities to support 3D, color graphics image rendering. All the enhancements are upwardly compatible with the 88100; that is, the 88110 can run existing 88100 binaries.

Base architecture extensions. We made the following minor enhancements of the base instruction set:

- Extensions of integer multiply and divide to improve support for signed multiplication and for arithmetic on higher precision integers. Instructions permit multiplication of two 32-bit numbers returning a full 64-bit result, and division of a 64-bit number by a 32-bit number, returning a 64-bit quotient.
- Additional information returned from the integer compare instruction to improve string-handling capability.
- Addition of static branch prediction to the branch opcodes, providing a mechanism by which the compiler gives the hardware a hint as to the direction a given conditional branch is likely to go. We estimate that the compiler potentially can statically predict more than 85 percent of the dynamically executed conditional branches correctly. We also believe that runtime branch profiling can further improve this rate in specific cases.
- An option that allows 16-bit immediate address offsets and literal constants to be treated as signed numbers (the 88100 treats them only as unsigned).

Floating-point architecture extensions. Our enhancements of the floating-point architecture were more significant. Anticipating heavy use of the processor for graphics and DSP and greater use of floating-point data in many general-purpose PC applications, we added the following:

- An extended floating-point register file to provide register name space for floating-point variables and frequently accessed constants beyond that provided in the 88100 architecture. The extended register file contains thirty-two 80-bit registers. Each register can hold one floating-point number of any precision—single, double, or double-extended. For compatibility with existing code, single- and double-precision floating-point numbers continue to be supported in the general register file as well. The compiler can use the additional register name space to improve code schedules and reduce memory references. This feature alone results in a speedup of more than 15 percent on the SPEC (Systems Performance Evaluation Cooperative) floating-point benchmarks.[7] The speedup is substantially greater on many graphics and DSP-intensive routines. We expect further improvement as compilers learn to take better advantage of this feature.
- Hardware support for IEEE-754, 80-bit, double-extended precision data,[8] to improve the accuracy and robustness of intermediate calculations in floating-point libraries.
- Hardware support for arithmetic on infinities, to eliminate the need for trapping to an IEEE software envelope to handle infinities, a frequent occurrence in some graphics algorithms.
- A time-critical floating-point mode to facilitate implementation of real-time DSP algorithms. In this mode, the hardware attempts to deliver an arithmetically sensible result rather than trapping on exceptional conditions such as underflow, overflow, and NAN (not a number).[8] For example, the hardware flushes underflows to zero rather than trapping to generate the exact IEEE-specified denormalized result. This feature is useful in real-time

algorithms because it reduces execution time and eliminates data-dependent time variations, thereby increasing the amount of work that can be scheduled up to a given deadline.

Graphics architecture extension. Fast processing of 3D graphics viewing transforms and lighting calculations requires high floating-point performance. However, good floating-point performance alone is not sufficient for good graphics performance. Due to the large amounts of data involved, graphics images are usually represented and rendered in packed, low-precision, fixed-point formats. Conventional microprocessors are not well suited to processing these data types. The traditional solution of adding a special-purpose coprocessor to the system increases costs and creates the difficulty of seamlessly integrating another processor architecture into the software environment.

A new set of instructions gives the 88110 this graphics capability. The hardware to implement these instructions takes only a small incremental investment in silicon (approximately 2.5 percent), while substantially increasing performance on fixed-point shading and image processing. For many systems, these instructions eliminate the need for coprocessors or special-purpose external hardware. For systems demanding greater graphics performance, a dual-88110 system provides the coarse-grain parallelism of a coprocessor approach yet preserves a homogeneous programming and software environment.

The new graphics instructions accelerate operations on the fixed-point and integer data types (Figure 3a,b) found in many 3D, color image-rendering algorithms. They operate on these packed data types 64 bits at a time.

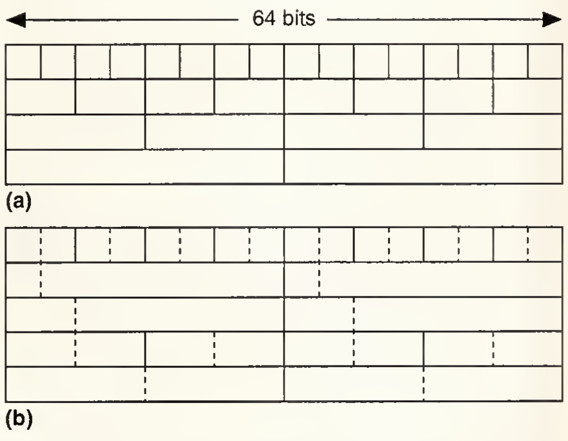


Figure 3. Graphics data formats: Packed integer data formats in pixels (a) and packed fixed-point data formats in color intensity values (b).

```
. padd.t     rD,rS1,rS2     ; add fields
. padds.x.t  rD,rS1,rS2     ; add fields with saturation
. psub.t     rD,rS1,rS2     ; subtract fields
. psubs.x.t  rD,rS1,rS2     ; sub fields with saturation
. punpk.t    rD,rS1         ; pixel unpack
. ppack.r.t  rD,rS1,rS2     ; pixel pack
. pmul       rD,rS1,rS2     ; multiply
. prot       rD,rS1,rS2     ; rotate
. prot       rD,rS1,<O5>    ; rotate immediate
. pcmp       rD,rS1,rS2     ; Z compare
```

Figure 4. Graphics instruction set.

The graphics instruction set (Figure 4) provides addition and subtraction on 8-, 16-, and 32-bit fields within 64-bit operands using either modulo or saturation arithmetic. Saturation arithmetic allows overflows or underflows within a field to clamp at the maximum or minimum value representable in the field rather than wrapping around as in normal modulo arithmetic. This method can be useful, for example, when addition of a color intensity value could result in an overflow that, in modulo arithmetic, would alias to a lower intensity value and thus produce an undesirable visual anomaly. Saturation is available on signed, unsigned, and mixed-sign numbers.

The set includes instructions for unpacking, truncating, packing, and rotating 4-, 8-, 16-, and 32-bit data fields to quickly convert between packed, fixed-point formats (intensity values) and packed, short, integer formats (pixels).

A graphics multiply instruction supports image-processing and -compositing algorithms, and a 64-bit compare instruction allows comparison of two pairs of 32-bit, fixed-point or floating-point *Z*-buffer values in one instruction.

Figure 5 (on the next page) is an example of how these primitive instructions can be chained together to implement complex image-processing operations such as compositing. A four-instruction sequence is illustrated. 1) The *punpk* instruction unpacks a 32-bit, four-channel, true-color pixel into four 16-bit, zero-padded, fixed-point numbers. 2) *Pmul* multiplies the result by an 8-bit integer, producing four new 16-bit, fixed-point numbers. 3) *Padd* adds these results to four other 16-bit, fixed-point numbers. 4) *Ppack* truncates the four 16-bit results of the addition to 8 bits, packs them together, and accumulates them with the pixel computed in the previous iteration of the loop by shifting the old pixel to the left and inserting the new pixel in its place. The program then can write this two-pixel result to the image buffer in memory, using a double-word store (St.d).

An important characteristic of the graphics instructions is their clean integration with the 88000 architecture, made possible by the 88000's special-function unit concept, which allows the instruction set architecture to be easily extended.

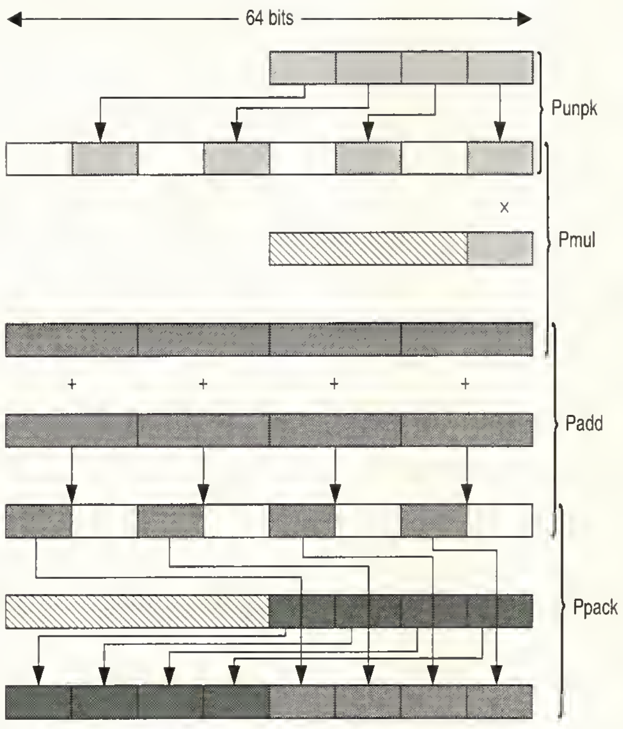


Figure 5. Graphics instruction chaining.

All the new instructions comply with the RISC philosophy of instruction set design.[9,10] They all take two 64-bit operands from the general register file, perform a simple operation, and produce a single 64-bit result. No instruction side-effects or special-purpose "kludge registers" are introduced into the programming model. Since data is kept in the general register file, all the existing 88000 arithmetic, logical, and bit-field instructions can be freely applied to the graphics data. Furthermore, the storage of data in the general register file allows the fixed-point graphics operations to overlap floating-point graphics operations, creating a high-throughput graphics pipeline.

Instruction fetch and issue

The heart of the 88110 microarchitecture is a centralized instruction sequencer, which dispatches instructions into an array of parallel execution units (Figure 6). The sequencer fetches instructions from memory, tracks resource availability and interinstruction dependencies, directs operand flow between the register files and execution units, and dispatches instructions to the individual execution units.

On each clock cycle, the sequencer fetches two instructions from the instruction cache and two from the branch target instruction cache. It decodes the appropriate instruction pair while fetching the necessary data operands from the register files. If all the required execution units and operands are available, the sequencer simultaneously dispatches both instructions to their respective execution units.

Instructions leave the sequencer in strict program order. The sequencer always tries to dispatch two instructions; if it can't, it tries to dispatch at least the first of the pair. In that case, the second instruction moves into the first issue slot, a new instruction is fetched to replace it, and the new instruction pair tries to issue on the next clock cycle.

Although the sequencer always dispatches instructions in order, not all instructions *issue*, or begin execution, in order. Reservation stations[11,12] in their respective execution units allow branches and stores to be dispatched even if their source operands are not available, so that further instruction dispatch can continue. Branch and store instructions wait in the reservation stations until the required source operands become available and the instructions can issue. Thus, branches and stores may issue out of order. This dynamic rescheduling[13] ensures that branches and stores, which normally constitute about 30-40 percent of the dynamic instruction mix, rarely stall on data dependencies and do not delay the dispatch of subsequent instructions.

Once the sequencer has dispatched an instruction into an execution unit pipeline, the instruction proceeds at a pace set by the capability of that unit. When an execution unit finishes an instruction, the sequencer controls write-back of the results into the register file and forwarding of the results to any execution unit that needs them immediately. The sequencer ensures that no register conflicts exist, but it is otherwise free to update the register file out of program sequence. This out-of-order instruction completion model allows useful work to proceed under long-latency operations. It also al-

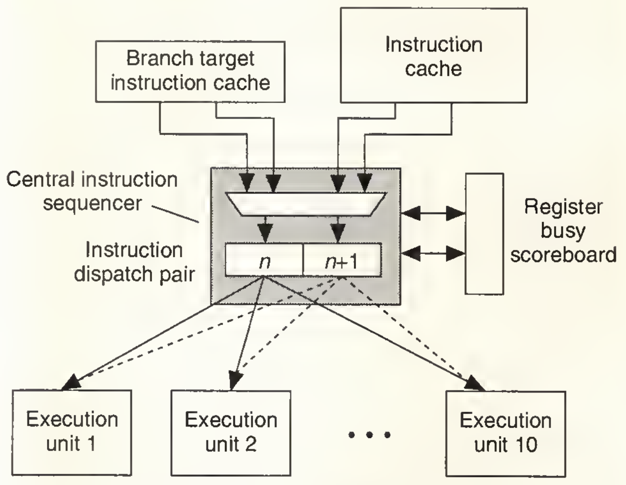


Figure 6. Instruction dispatch.

lows a mixture of execution unit types, possibly with variable-length pipelines, without resorting to a common, long, fixed-length pipeline with complicated pipeline bypass circuitry.

The master instruction pipeline, shown in Figure 7, is a conventional, four-stage RISC pipeline that completes most instructions in three clock cycles. In the first stage of the pipeline, the sequencer fetches an instruction pair from the instruction cache. In the second stage, it decodes these two instructions, fetches their operands from the register files, and decides whether or not to dispatch them into execution. Instructions execute during the third stage; for most instructions the execute stage requires one clock cycle, but some take more. In the fourth and final stage, the sequencer writes the results from the execution units into the register files.

Three things can prevent an instruction from issuing: 1) A necessary resource is not available or is busy (structural hazard); 2) an operand conflict exists with a prior instruction (data hazard); or 3) a branch causes a change in program flow, requiring an alternate instruction stream to be fetched, thus temporarily starving the dispatch unit of instructions (control hazard).[4]

Structural hazards. Structural hazards occur because of pipeline resource or instruction class conflicts. Pipeline conflicts are rare in the 88110 because the register files are multiported with full-width data paths, and all execution units (except the divider) either execute in one cycle or are fully pipelined to accept a new instruction each clock cycle.

Instruction class conflicts occur when two instructions requiring the same execution unit attempt to issue on the same clock cycle; for example, two multiply instructions attempt to issue as a pair, but only one multiplier execution unit exists. The concurrency matrix in Figure 8 on the next page shows the relatively few pairings of instructions in the 88110 that will stall due to a class conflict. We eliminated a significant number of class conflicts by providing a duplicate set of integer ALUs (arithmetic logic units).

An important aspect of the 88110's superscalar instruction issue capability is that it is symmetrical; that is, any instruction can be dispatched from either slot in an instruction dis-

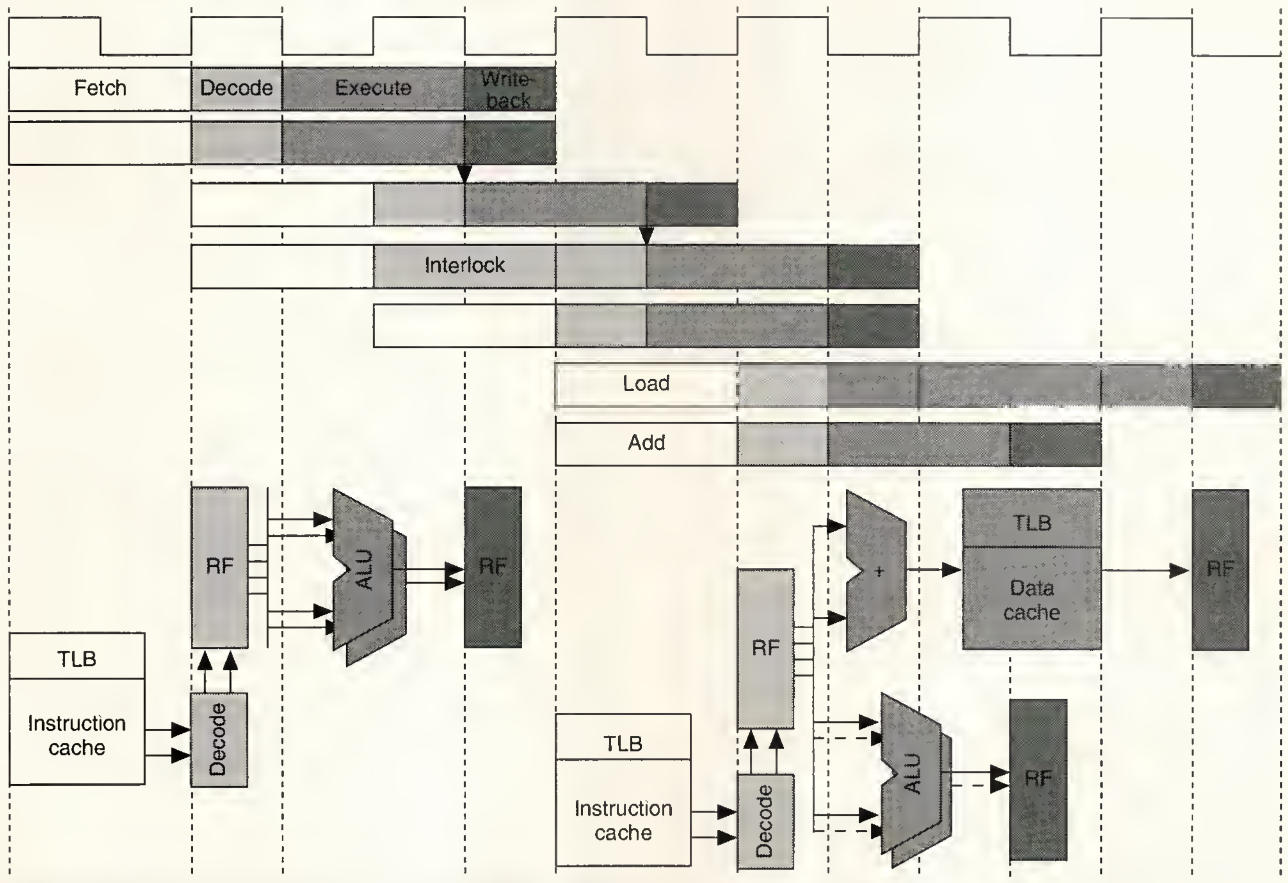


Figure 7. Master instruction pipeline; RF indicates register file.

n \ n+1	Arithmetic	Logical	Bit field	Multiply	Divide	Branch	Load	Store	Floating-point multiply	Floating-point add/sub/cmp	Floating-point divide	Graphics multiply	Graphics add & sub	Graphics packing
Arithmetic	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Logical	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Bit field	▨	▨		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Multiply	▨	▨	▨		▨	▨	▨	▨		▨	▨		▨	▨
Divide	▨	▨	▨	▨		▨	▨	▨	▨	▨		▨	▨	▨
Branch	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	▨
Load	▨	▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨
Store	▨	▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨
Floating-point multiply	▨	▨	▨		▨	▨	▨	▨		▨	▨		▨	▨
Floating-point add/sub/cmp	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨
Floating-point divide	▨	▨	▨	▨		▨	▨	▨	▨	▨		▨	▨	▨
Graphics multiply	▨	▨	▨		▨	▨	▨	▨		▨	▨		▨	▨
Graphics add & sub	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨
Graphics packing	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	

▨	Can issue simultaneously
Cmp	Compare

Figure 8. Instruction concurrency matrix.

patch pair, as illustrated in Figure 9. For example, the sequencer can dispatch a multiply instruction to the multiplier regardless of whether the instruction is in the first or second slot of a dispatch pair. Thus, the 88110 has none of the artificial instruction ordering or pairing restrictions characteristic of VLIW or restricted superscalar machines. Also, the sequencer fetches instructions from the instruction cache two at a time regardless of their address alignment, so no alignment restrictions must be met. Removing these constraints frees the compiler to optimize for more important considerations.

Data hazards. Instruction issue can also stall because of data hazards such as read-after-write (true data dependency), write-after-write (output dependency), or write-after-read (antidependency) hazards. Read-after-write hazards occur when an instruction needs a result from a previous instruction that has not yet completed execution. Write-after-write hazards occur when an instruction writes to a register after a subsequent instruction has already written to the same register, thus leaving the register with old data. Write-after-read hazards occur if an instruction attempts to write a result to a register before a previous instruction reads the old value. The write-after-write and write-after-read hazards are really false dependencies, since they involve no true data dependency, only a register name conflict.

A register-busy scoreboard automatically interlocks the 88110 pipeline against incorrect data on hazards by tracking source and destination operand availability. Each time the sequencer dispatches an instruction, it also marks the instruction's destination register as busy until the instruction completes execution. As the sequencer considers instructions for dispatch, it checks the scoreboard to ensure that no register conflicts exist with prior instructions still in execution. (The term *scoreboard*, as applied to computers, originally referred to the complex centralized queue and reservation mechanism used in the CDC 6600 for tracking all aspects of out-of-order execution.[14] Recently, however, the term has become generic, referring to any control unit that handles register reservations[12]—including much less sophisticated units than the CDC 6600's—such as that in the 88110.)

The sequencer avoids incorrect data on read-after-write hazards by checking the source operand register scoreboard bits and on write-after-write and on write-after-read hazards by checking the destination operand register scoreboard bit. The sequencer can dispatch branches and stores even if their source operand is busy. However, they are held in a reservation station and do not begin execution until the scoreboard

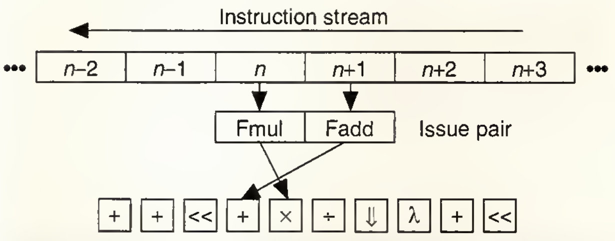


Figure 9. Symmetrical superscalar instruction issue.

bit for the needed register clears and the operand can be read.

Of the three types of data hazards, read-after-write hazards cause the most instruction issue stalls in the 88110. We keep these stalls short by providing 1) low-latency execution units and 2) a register file bypass from the execution unit result buses to the execution unit inputs. The bypass makes instruction results available immediately to subsequent instructions without having to wait an extra clock cycle for results to be written into and read out of the register file.

For the most part, the 88110 relies on static scheduling of the instruction stream to avoid stalling on hazards. In many cases, static scheduling is straightforward and the compiler can handle it effectively. However, statically scheduling code around some types of data hazards can be difficult or inefficient; that is why the 88110 performs dynamic rescheduling of branches and stores.

As an example of dynamic rescheduling of stores, consider the common operation of fetching data from memory, performing some computation on it, and then storing the result back in memory. If the computation requires multiple cycles—as a floating-point multiply does, for example—the store of the result introduces a data hazard that would stall instruction issue even though no further need exists for that data in the program. The store reservation stations allow stores to be set aside and instruction issue to continue while the store data is being computed. Then, when the store data becomes available, the sequencer immediately forwards it to the appropriate reservation station and allows execution of the store to begin.

Similar stalls could occur on conditional branch operand data hazards—due either to long-latency operations (such as load-branch sequences) or to dispatch pair dependencies (such as compare-branch pairs). As with stores, the 88110 provides a reservation station to avoid stalling on these branches. In the case of branches however, an additional problem exists. The machine does not know where to continue execution until the branch operand is available and the sequencer can evaluate the condition. Therefore, the sequencer predicts the branch direction, and instructions down the predicted path execute conditionally, or speculatively, until the branch operand is resolved. The static prediction of the branch direction is based on the opcode of the branch instruction.

The branch reservation station provides a place to set aside the branch instruction so that instruction issue can continue while the branch condition is being resolved. Once the operand becomes available and the condition is evaluated, the machine determines whether or not instruction execution actually went down the correct path. If it did, useful work was accomplished and execution simply continues uninterrupted. If the prediction was incorrect and execution went the wrong way, the machine backs up to the branch, undoing all changes made to the registers by conditionally executed instructions, and resumes execution down the other path.

Figure 10 contrasts the pipeline situation with and without speculative execution on a taken conditional branch (bcnd) that is dependent on a load. With speculative execution and branch prediction (top), the new instruction stream (target 0, target 1, and so on) begins execution immediately with no bubbles introduced into the pipeline. (By *bubbles* we mean lost opportunities for instructions to issue.) Without speculative execution and branch prediction (bottom), the machine would continue fetching down the sequential instruction stream (next 0, next 1), since the target address would not yet be available. Also, instruction dispatch down the target

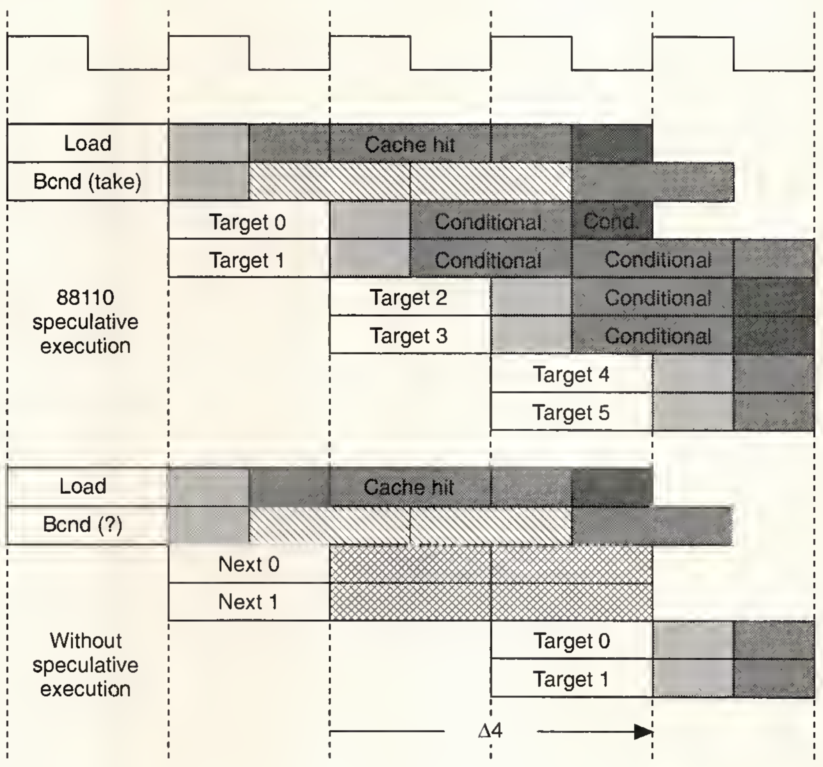


Figure 10. Speculative execution.

path would have to delay for two clock cycles (four bubbles) while the machine waits for load data with which to compute the branch direction.

During speculative execution, the instruction fetches that miss the instruction cache access the bus just as in normal execution. Load instructions can access the data cache on a hit, but the bus will not service a data cache miss until the branch condition resolves. Store instructions can be dispatched to the reservation stations but can never access the cache or the bus during conditional execution. This procedure prevents corruption of the memory image by a store instruction that is eventually canceled on a misprediction.

The accuracy of branch prediction and the penalty for mispredicting are important, of course, to the overall performance gain realized from speculative execution. Although prediction accuracy depends on the compiler being used and the nature of the application, our simulations indicate that good static-prediction accuracy is achievable. On the SPEC benchmarks over 80 percent of all conditional branches take the anticipated path, and over 70 percent of the branches that need to be predicted are being predicted correctly. Currently, our compiler predicts only the simple branch cases, so we expect these results to improve as the compiler becomes more aggressive. Also, we are currently seeing a penalty of less than one-half percent for mispredicting branches, although the penalty may increase on applications that allow deeper speculative execution.

Control hazards. Generally speaking, when a pipelined processor encounters a branch, it needs time to evaluate the condition, compute the target branch address, fetch instructions from the new target instruction stream, and refill the pipeline. The 88110 deals with pipeline bubbles caused by control hazards by means of the speculative execution model just described and by the use of a branch target buffer to shorten branch execution latency.[15]

The speculative execution model permits out-of-order instruction execution to extend beyond the domain of a basic block. It also helps keep the instruction pipeline full and the execution units busy even in the face of small basic blocks, but it does not address branch latency.

RISC designers traditionally compensated for branch latency by using a branch-and-execute[9] or delayed-branch[16] strategy to give the processor something to do while the branch executes. In a superscalar design, however, a single architectural delay slot is insufficient to cover the two instruction bubbles inserted into the instruction pipeline by each clock cycle of branch latency. Short branch latency is important, even with speculative execution, to minimize the number of instructions subject to cancellation in the event of misprediction.

Due to the critical importance of control hazards to performance, we invested a significant amount of circuitry in the 88110 to reduce branch latency. During the instruction decode phase of the pipeline, the sequencer fetches two instructions at the branch target address from the branch target instruction cache (TIC) and supplies them as the first two instructions down the branch-taken path. The sequencer evaluates (or predicts, if necessary) the branch condition early in the pipeline to select either the target instruction pair from the TIC or the next sequential instruction pair from the instruction cache in time for the next instruction decode phase. By this time, with the branch target address computed, the instruction cache can supply further instructions along the target instruction stream. Thus, on a hit, the TIC can fill the two branch pipeline delay slots with useful instructions.

The TIC has 32 entries and is fully associative. Each entry in the TIC holds the first two instructions from a recently taken branch target path. The hardware automatically loads cache entries on a miss, following a FIFO replacement policy. The sequencer uses the logical address of the branch being evaluated to index the TIC. Thus, the target instructions are available immediately for the next instruction fetch phase of the pipeline.

Even with its heavily pipelined, out-of-order execution model, the 88110 implements fully precise exceptions.

TIC hit rates depend heavily on the application, but our simulations show an average TIC hit rate of around 85 percent on the SPEC benchmark suite.

Exceptions

Program exceptions and interrupts have always been a problem in pipelined machines, especially those that execute instructions out of order and/or speculatively. During normal program execution, machine state changes can occur out of program sequence as long as the state currently relevant to the program *appears* to be in the correct program order. But in a parallel machine, the internal pipelines and buffers temporarily hold much of the dynamic machine state. Thus, at any point in time, the register files do not completely reflect the true current machine state. So, when a program exception occurs and the instantaneous state of all the registers become manifest, the dynamically held state can be lost or confused. The register file's inconsistency with the actual machine state makes correct recovery from the exception difficult or impossible.

Even with its heavily pipelined, out-of-order execution model, the 88110 implements fully precise exceptions. That is, the processor always presents the architecturally correct state to an exception-handling routine. It also gives an exact indication of which instruction caused the fault and where to resume execution. The 88110's precise exception model dramatically simplifies and speeds up exception- and interrupt-handling software routines.

When an instruction generates an exception—for example, a page fault or an arithmetic overflow—instruction execution continues until all instructions that issued prior to the faulting instruction complete. (This step ensures that all synchronous exceptions occur in strict program order). At this point, execution stops, the internal pipelines are cleaned up, and the machine backs up to the instruction that caused the exception, leaving all registers in the precise architectural state that existed before the faulting instruction issued. The 88110 accomplishes this by means of a history buffer,[17] which records the relevant, user-visible machine state as instructions issue. The processor uses information stored in the history buffer to quickly restore the machine state back to the point of the exception. This is the same mechanism the 88110 uses to recover from mispredicted branches.

When the machine recognizes an asynchronous external interrupt, it halts execution, aborts all unfinished instructions (or waives write-back in the case of a memory transaction in progress on the bus), and backs out the effects of any instructions that completed out of order. This procedure minimizes interrupt response latency, a critical parameter in many real-time system applications.

Register files

The 88110 has two sets of register files, the formats of which are shown in Figure 11a,b. The general register file primarily holds fixed-point values and address pointers; the extended register file holds floating-point data.

The general register file has thirty-two 32-bit registers, which can be used by all instructions in the machine. These registers are accessible in pairs to supply 64-bit operands whenever necessary—for example, for graphics and double-precision floating-point values. Register zero is hardwired to the integer constant zero (0).

The extended register file is a new addition to the original 88000 architecture. It contains thirty-two 80-bit registers, which are used exclusively by the floating-point instructions. Each extended register can hold one floating-point number in either single, double, or double-extended format. Register zero in the extended register file is hardwired to the floating-point constant positive zero (+0.0E00). We provided instructions that quickly move data back and forth between the two register files.

Both eight-ported register files can supply all the operand bandwidth required to sustain the peak instruction issue rate of two instructions each clock cycle, regardless of the instruction mix or data precision. Data-forwarding paths around

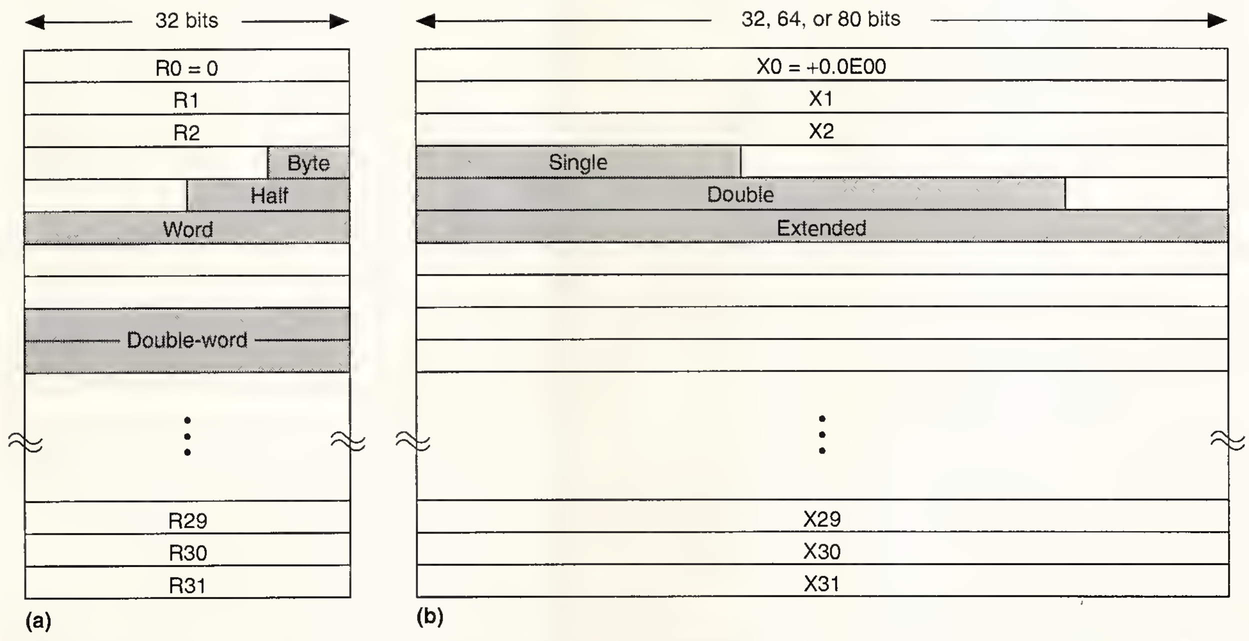


Figure 11. Register files: General (a) and extended or floating-point (b).

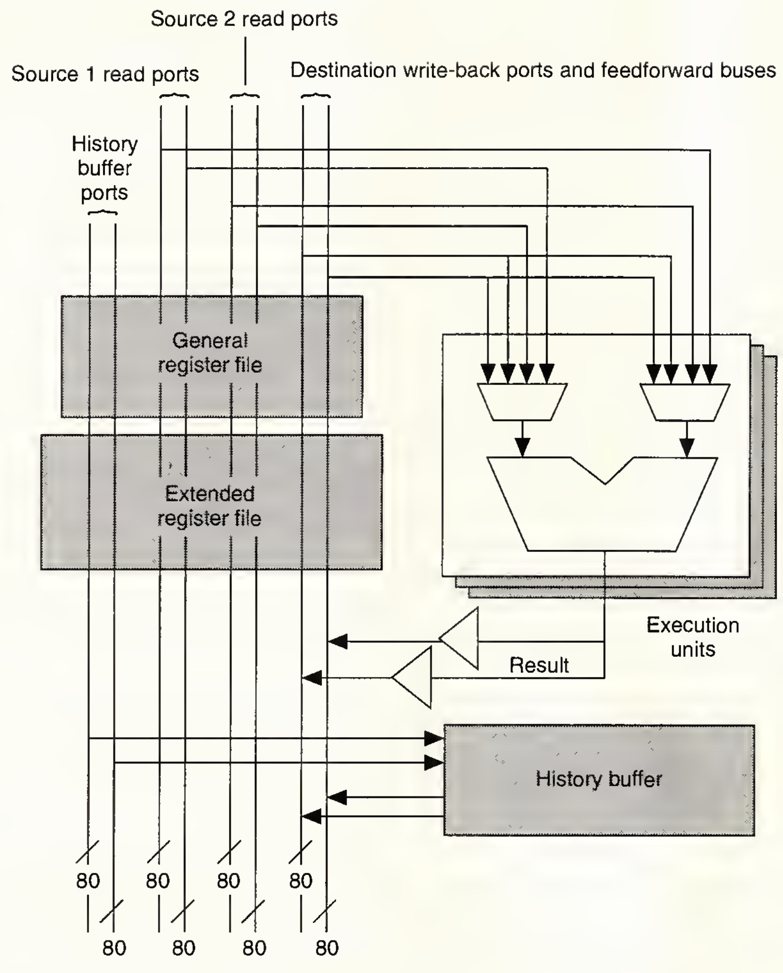


Figure 12. Operand data paths.

the register files route a result returning from an execution unit directly to the inputs of a waiting execution unit while the result is also being written into the register file (see Figure 12). This approach avoids stalling the instruction issue an extra clock cycle while data is written into the register before it can be read out on a source port.

Execution units

The 88110 contains 10 independent execution units: branch, integer (two), bit field, multiplier, floating-point adder, divider, graphics (two), and data or load/store. The dataflow paths from outside the chip, through the caches and register files, and into and out of the execution units, are shown schematically in Figure 13.

Integer units. The integer units are simple 32-bit ALUs that handle all the fixed-point arithmetic instructions and logical instructions. The execution latency of both integer units is one clock cycle.

Bit field unit. This unit is a shifter/masker circuit that handles the 88000's extensive set of bit-field manipulation instructions. It also has a single-clock-cycle execution latency.

Multiplier unit. The multiplier unit handles all 32- and 64-bit signed and unsigned integer multiplies, the graphics multiply, and the single-, double-, and extended-precision floating-point multiplies. The fully pipelined unit can start a new multiply instruction every clock cycle. The multiplier has an execution latency of three clock cycles for all data types. The 32 × 64-bit multiplier uses Booth partial product generators and a Wallace tree to sum the partial products twice each clock cycle to maximize circuit efficiency.

Floating-point adder unit. The floating-point adder executes all single-, double-, and extended-precision floating-point add, subtract, compare, and integer conversion instructions. The fully pipelined unit can start a new instruction on every clock cycle. The adder has a three-clock-cycle execution latency for all precisions. A special shortcut reduces the latency for floating-point compare to one clock cycle. The dynamic 64-bit adder circuit uses a combined block-carry-look-ahead and fast-carry-select scheme. The actual 64-bit add time is much shorter than one clock cycle. Most of the three-clock-cycle execution time occurs because of the floating-point format operations such as reserved-operand check, exponent debiasing, mantissa alignment, normalization, and rounding.

The construction of a fully pipelined floating-point multiplier and adder with short latencies is very hardware intensive. The return on this investment is the ability to achieve a more efficient static code schedule and an extremely high floating-point throughput. Long-latency operations require that a large number of independent instructions be available to be scheduled into the pipeline delay slots to avoid bubbles and keep execution unit usage high. In general, the compiler needs to find less program parallelism on hardware with short latencies than it does on hardware with long latencies to achieve the same level of performance.

One commonly used technique for scheduling around long-latency operations is loop unrolling. This technique increases the basic block size and the number of data-independent operations available to be scheduled into pipeline delay slots. A difficulty with loop unrolling is that it requires a large reg-

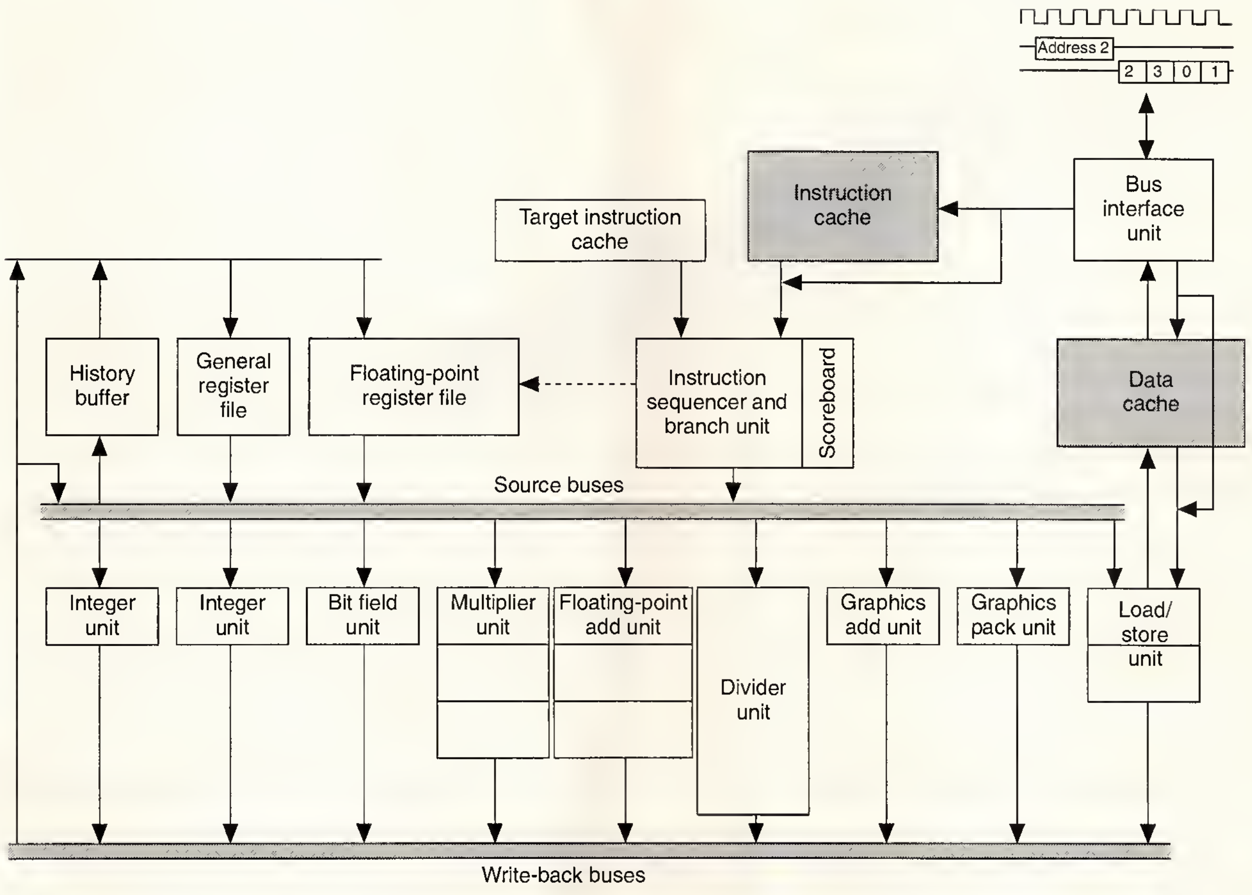


Figure 13. Organization of the 88110.

ister name space so registers can be allocated to avoid data hazards. It also increases the static code size, which can waste memory space and instruction cache entries. The 88110's short, three-cycle floating-point latency is well balanced with the large register files, making a small amount of loop unrolling effective without requiring elaborate register-renaming hardware.[11]

Divider unit. The divider handles all 32- and 64-bit signed and unsigned integer divides and all single-, double-, and extended-precision floating-point divides. The iterative divider uses a radix-8-per-clock SRT algorithm (Sweeney-Robertson-Tosher) with a latency dependent on the operand type and precision. The execution latency of single-precision floating-point division equals 13 clock cycles.

Graphics units. Two execution units implement the new graphics instructions. One handles the arithmetic operations, and the other handles the bit-field packing and unpacking instructions. Both units have a single-clock-cycle execution latency. Because the two units are independent, each can accept a new instruction every clock cycle. In fact, the instructions are partitioned in a manner that often makes it possible to schedule graphics algorithms to sustain a throughput of a full two instructions each clock cycle. As an example, Figure 14 on the next page shows execution of the inner loop of a simple Gouraud shading algorithm. Since the graphics units behave the same as other execution units, graphics instructions can issue together with any other integer, floating-point, or memory-referencing instruction. This flexibility minimizes loop overhead and allows very efficiently scheduled graphics routines.

Load/store unit. The load/store unit is the most sophisticated execution unit in the 88110. We invested a considerable amount of circuitry in this unit because of the critical importance of memory referencing to overall performance. The unit provides a stunt box[14] capability for holding memory references that are waiting for the memory system and allows dynamic reordering of loads past stalled stores. (The CDC 6600's stunt box did not allow loads to pass stores, but the term *stunt box* has come to refer to any device that allows reordering of memory references in the memory system.[12])

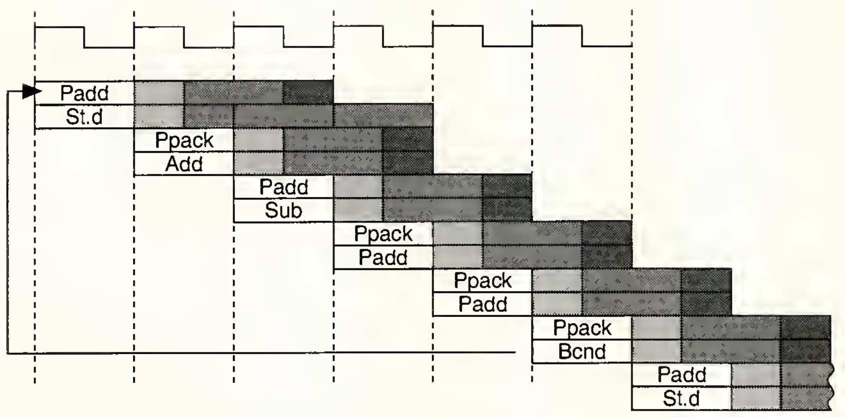


Figure 14. Graphics execution unit parallelism.

The load/store unit executes all instructions that transfer data between the data cache, or bus interface, and the register files. The data path from the load/store unit to the register files is a full 80-bits wide. Load latency for 32- and 64-bit data on a cache hit is two clock cycles—one longer than a normal integer add instruction.

On each clock cycle, the unit can accept one new load or store instruction from the instruction dispatch unit. When an instruction is dispatched to the load/store unit, it awaits access to the data cache in either the load queue or the store queue[12] (see Figure 15). Normal instruction dispatch and execution can continue while these instructions await service by the cache or memory system. On properly scheduled code, this buffering provides considerable tolerance of long memory latency.

The load queue is a simple, four-deep FIFO queue. The store queue is a somewhat more complex three-deep reservation station that is also managed as a FIFO queue. Since store instructions can be dispatched before the store data operand is available, store instructions wait in the store queue until the instruction computing the required data completes execution. When the operand becomes available, the sequencer directs it into the store reservation station, and the associated store instruction becomes a candidate for access to the data cache.

If a store instruction stalls in the reservation station waiting for its operand, subsequently issued load instructions can bypass the store and immediately access the cache. An address comparator detects address hazards and prevents loads from going ahead of stores to the same address, thus getting stale data. This load/store reordering feature allows runtime overlapping of tight loops by permitting loads at the top of a loop to proceed without having to wait for the completion of stores from the bottom of the previous iteration of the loop.

The data cache is nonblocking, or lock-up free,[18] for store-load accesses. For example, when a load bypasses a store and misses the cache, the cache can be decoupled from the bus so that the store can access the cache while the bus waits for memory. This is also true for a store miss followed by a load hit and for the user-mode touch-load instruction.

Touch-load provides a limited form of decoupling of load-store and load-load sequences. This instruction provides a mechanism for a program to bring a cache line into the cache before it is actually needed. While the load/store unit waits for memory to deliver the data, instruction issue can continue unrestricted. During that time, load and store instructions can access the cache. The programmer can use the touch instruction to prefetch data into the cache to avoid load misses that would likely stall execution if serviced on demand. When used properly, these instructions can significantly increase cache hit rates and minimize load miss penalties.

The load/store unit implements other user-mode instructions to allow more effective scheduling of the data cache. An allocate instruction allocates a line in the cache without first bringing the line from memory. A program can use this instruction to avoid unnecessary bus transactions in cases where it will overwrite the entire cache line anyway. In addition, a line flush instruction can force a cache line out to memory. The line flush provides a mechanism to update a video frame buffer without allocating the frame buffer as write-through storage and thereby sacrificing the burst-mode line transfer capability of the bus.

The load/store unit also contains a selective cache bypass[19] capability for stores. Store instructions can be selectively marked (with an opcode bit) to "store-through" the cache. Such a store bypasses the cache and proceeds directly to memory. If the reference hits the cache, the cache is updated; but if it misses the cache, no new line is allocated. This feature prevents pollution of the cache with data known to be of no further use to the program. It can increase cache hit rates by avoiding replacement of more useful entries in the cache. It can also improve cache miss latency by reducing the number of "dirty" lines that have to be copied back to memory when a new cache line is allocated.

The load/store unit also executes the 88000's XMEM instruction, which performs an atomic read-write operation that exchanges the contents of a register with a memory location. A program can use this instruction to implement semaphores and shared-resource locks in multiprocessor systems.[20] It can be used as a primitive to construct a wide variety of complex synchronization protocols, such as spin-lock, compare-andswap, and fetch-and-add. For example, one can construct an efficient spin-lock by repeatedly polling a lock with loads, which will hit in the cache and therefore not generate bus traffic. When the current owner of the lock releases it, the cache coherence logic brings the new copy of the lock into the cache, and the processor can then try to acquire the lock with an XMEM. The resulting indivisible read-write bus transaction ensures exclusive ownership of the lock.

The 88000 architecture uses primarily a "big-endian" byte order—that is, an address points to the most significant byte of a datum in memory—as opposed to a "little-endian" order—in which an address points to the least significant byte of a datum. The 88110 provides a solution for heterogeneous big/little-endian multiprocessor systems with a mode switch that allows data memory references to be performed in either big- or little-endian fashion. In little-endian mode, the load/store unit swaps the bytes in all half-words, words, double-words, and quad-words as they transfer into or out of the cache.

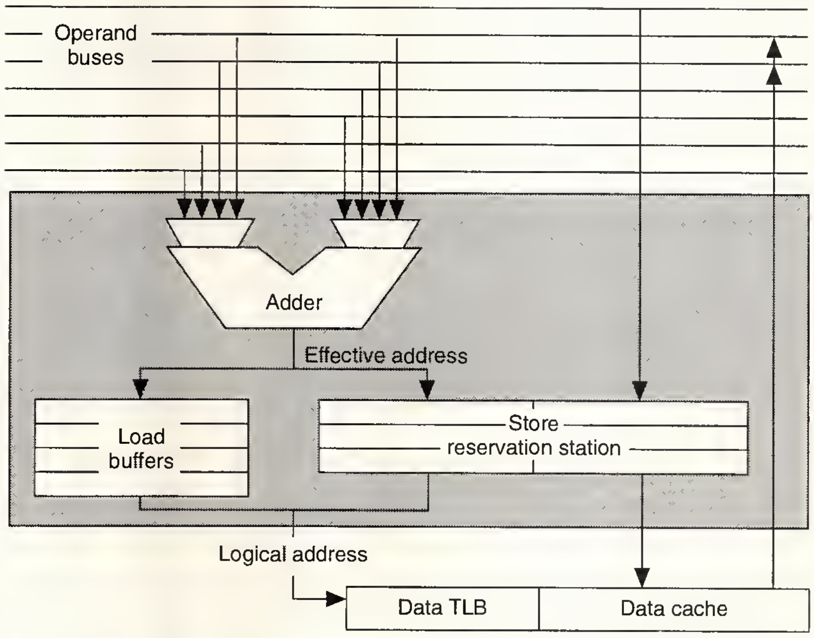


Figure 15. Load/store execution unit.

Address translation facilities

The 88110 offers full hardware support for a demand-paged virtual memory system.[21] It provides hardware facilities for translating logical effective program addresses to physical memory addresses, for protecting areas of memory from unprivileged accesses, and for trapping to supervisory routines on accesses to pages not currently in memory. The organization of the address translation facilities is diagrammed in Figure 16.

The processor contains two independent and concurrent translation look-aside buffers—one for translating instruction addresses, the other for data addresses. Each fully associative TLB can hold thirty-two 4-Kbyte page address translation entries. Each entry contains a translation descriptor that maps a virtual page to its corresponding physical page number.

On each memory reference, the hardware looks up the logical address (which is equal to the virtual address in the 88110) by simultaneously comparing it to all en-

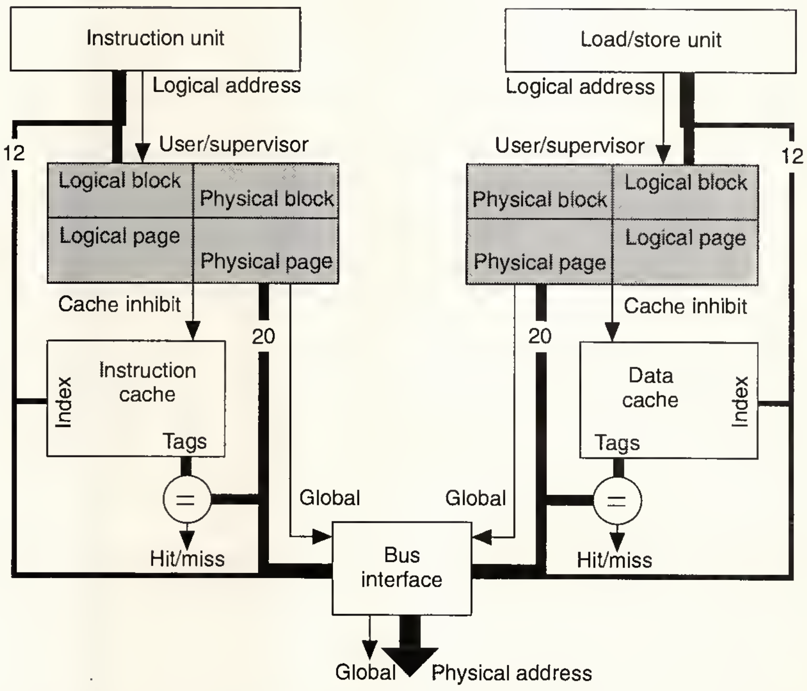


Figure 16. Virtual address translation facilities.

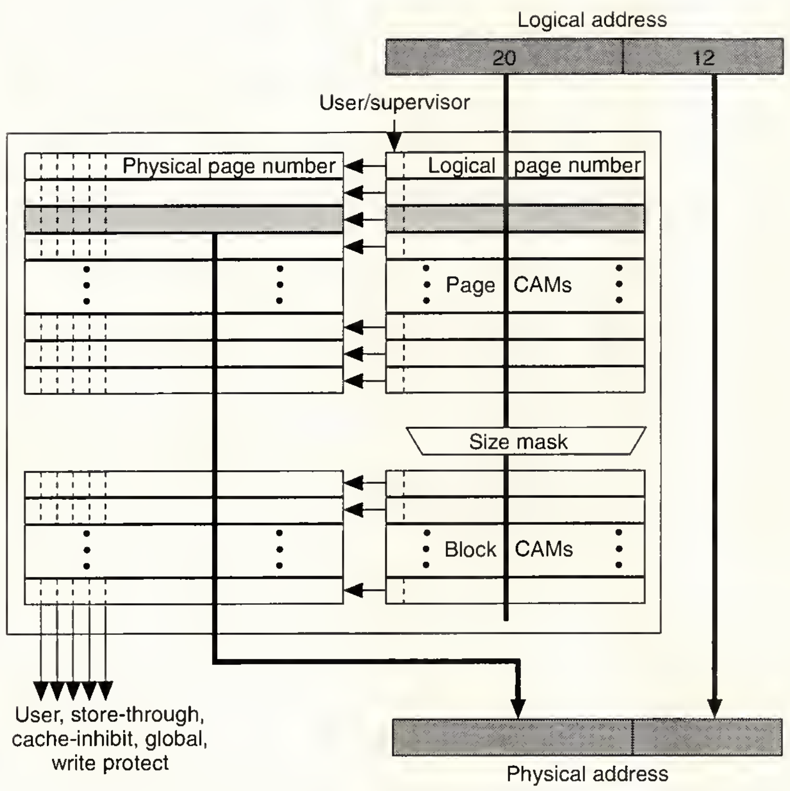


Figure 17. Translation look-aside buffer (TLB).

tries in the TLB, using content-addressable memory (CAM) elements as illustrated in Figure 17. Each CAM element is associated with one physical page descriptor, which is loaded into the TLB from the memory-based page tables maintained by the virtual memory operating system software.

If the TLB lookup finds an entry that matches the logical address of the memory reference being translated, it is a TLB hit and that entry is used to translate the address. If the memory reference does not violate the access privileges specified by the selected TLB entry, the page offset bits from the least significant 12 bits of the logical address form the final translated physical address. If the reference does violate the access privileges, the processor aborts the memory reference and signals an attempted memory protection violation to the operating system by taking an access exception trap.

Information stored in the TLB also governs certain caching policies, such as global access and cache bypass. The global property indicates that the referenced memory page is shared, and, therefore, other processors on the bus must "snoop" (watch for) any external bus transaction generated as a result of a cache miss to this page. The cache-inhibit and write-through properties allow data cache bypass to be controlled on an address basis at the granularity of a page. When a page is marked "cache-inhibited," all references—loads or stores to that page—bypass the cache. If a page is marked "write-through," all stores to that page bypass the cache. This write-through capability is similar to the selective cache bypass (store-through) feature described earlier, but it allows bypassing on an address (page) basis rather than an instruction basis.

If the memory reference matches an entry in the TLB (a hit) but the entry is marked "invalid," the hardware generates a page fault trap, and control transfers to a supervisory routine. This routine would bring the accessed page in from disk, update the system page tables, and reissue the faulting memory instruction.

If the memory reference does not match any entry in the TLB (a miss), one of two things can happen. The hardware can automatically walk through the operating system's page tables to load a new descriptor into the TLB before starting the memory transaction. Or, the hardware can generate an exception, invoking a software routine to manually load a new descriptor into the TLB and then rerun the faulting memory instruction. The software TLB-reloading mechanism provides the flexibility to support virtual memory management systems that use table structures (such as inverted page tables) different from those supported directly by hardware.

The simple but efficient hardware table-walking algorithm illustrated in Figure 18 indexes through two levels of system segment and page tables to locate a page descriptor, which is then brought into the TLB. The algorithm includes an indirection capability, which treats the resolved page descriptor as a pointer that is followed one additional level to a final page descriptor. Indirection allows multiple virtual addresses that map (alias) to the same physical address to be mapped through a common page descriptor. This capability simplifies system maintenance of the page referenced and modified status bits, typically used to implement an efficient demand-paged, virtual memory management system.

Each TLB implements facilities for the operating system to determine whether a particular translation is currently in the TLB, for locating and invalidating individual entries in the TLB, and for invalidating all user or supervisor entries. These facilities support many aspects of virtual memory management, including TLB coherence protocols such as the Mach operating system's TLB "shoot-down" algorithm.[22]

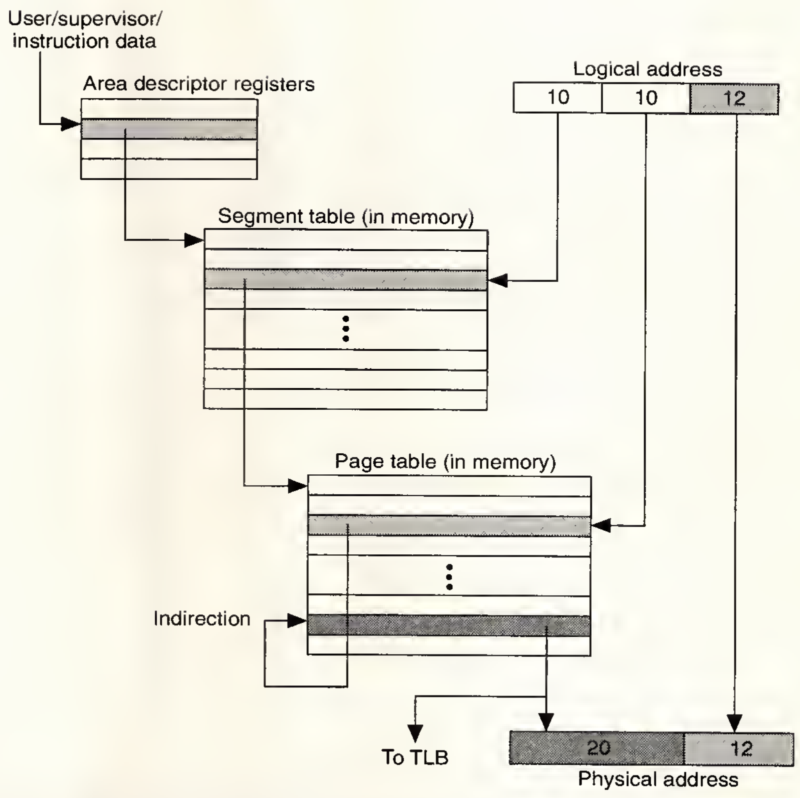
Another important feature of the TLBs is the block address translation facilities. In addition to the 32 page entries already described, each TLB has eight variable-size block address translation entries. The block translation entries perform address translation in a similar fashion to the page entries but are capable of mapping large blocks of memory (512 Kbytes to 64 Mbytes). These entries allow mapping of large static areas of system code or data structures and large entities such as frame buffers, without using an excessive number of TLB page entries.

Caches

The 88110 has a Harvard-style internal architecture—that is, it has separate, independent instruction and data paths. An on-chip instruction cache feeds the instruction unit, and an on-chip data cache feeds the load/store execution unit. Cache misses are multiplexed together and are serviced from a common external bus interface.

The instruction and data caches are both physically addressed. Physical caches have the advantage over logical caches in that synonyms do not occur. As a result, special precautions to disambiguate logical addresses across different process contexts are not necessary. No extra hardware is required for associating a logical address with a specific process, nor is it necessary to incur the overhead of flushing the caches on a context switch. Physically addressed caches also simplify maintenance of cache coherency in multiprocessor systems.

In the 88110 implementation, the caches are logically indexed and physically tagged, as illustrated in Figure 19. The cache arrays are directly indexed with the 12 untranslated page offset bits from the least significant portion of the logical address. Each cache line is tagged with the high-order 20 bits of the fully translated physical address. This arrangement allows selection of the cache set and retrieval of the cache tags and data in parallel with the translation of the logical address to a physical address in the TLB. After the physical address becomes



User/supervisor/ instruction data
Area descriptor registers
Logical address
10
10
12
Segment table (in memory)
Page table (in memory)
Indirection
To TLB
20
12
Physical address

Figure 18. Automatic hardware table-walking scheme.

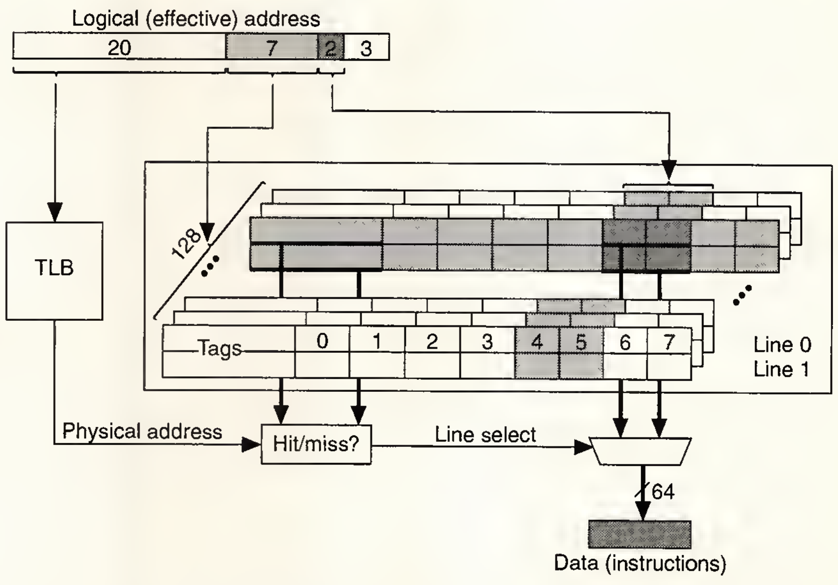


Figure 19. Organization of instruction and data caches.

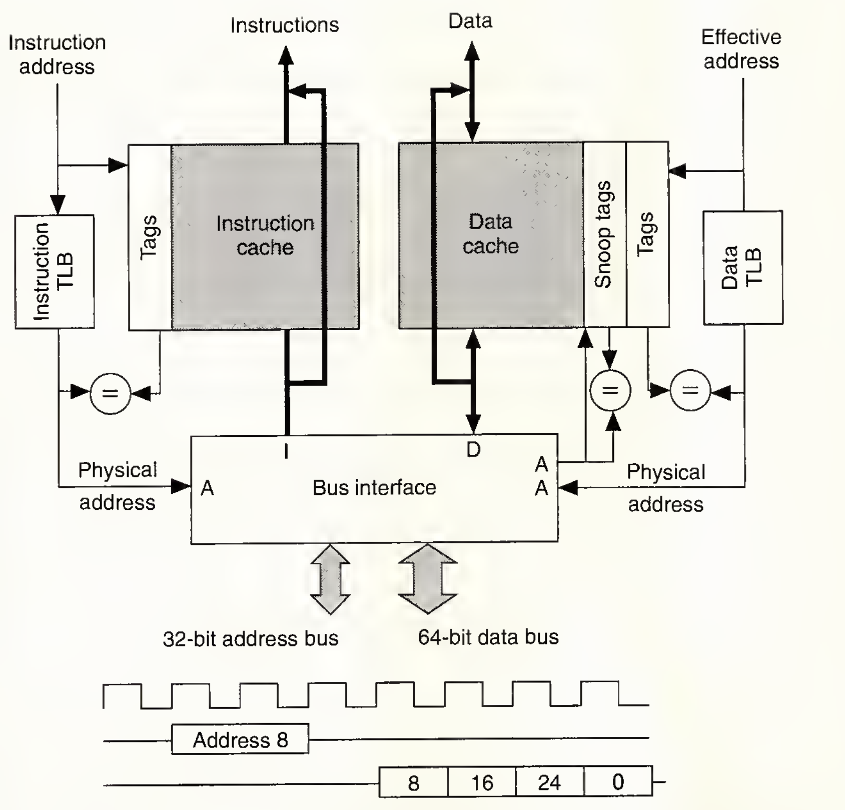


Figure 20. Cache miss.

available, the machine compares it against the two tags (one associated with each line of the selected cache set) to determine a hit or miss and make the final line selection.

Instruction cache. The 8-Kbyte, two-way, set-associative instruction cache is organized into 128 sets, with two lines for each set, and 32 bytes (eight instructions) in each line. The two-way set associativity gives the cache substantially better hit rates than direct-mapped caches at the 8-Kbyte size, and, due to the implementation techniques used, does not adversely affect clock cycle time. The cache has a one-clock-cycle access time and can provide a pair of instructions (64 bits) to the instruction dispatch unit on each cycle, regardless of whether the access is aligned to an odd or an even word address. The only time the cache fails to deliver two instructions is when the instruction pair straddles a cache-line boundary. In practice this turns out to have a very minor performance impact.

We designed the cache to minimize latency on a miss. Figure 20 shows the hardware involved in a cache miss. On a miss, the processor initiates an eight-word burst transaction on the bus to fill the cache line. The burst can transfer two instructions from the bus into the cache on each clock cycle. The burst begins with the missed instruction pair, continues transferring to the end of the cache line, and then wraps around to fill the beginning of the cache line (if necessary). As soon as the cache receives the missed instruction from the bus, it forwards the instruction directly to the instruction unit so that execution can resume immediately. As the cache receives subsequent instructions from the bus, it also streams them directly into the instruction unit so that execution doesn't stall while the cache line is being brought in from memory.

As shown in Figure 21, the cache is arranged in eight 1-Kbyte blocks; each block is 64-bits wide by 128 rows. Each row contains one 32-bit word from each of the two cache lines. Four of the cache blocks hold the even words of a pair; the other four hold the odd words. When access is made to an evenly aligned instruction pair, the least significant word returns on the even bus and the most significant word returns on the odd bus. When access is made to an oddly aligned pair, the least significant word returns on the odd bus and the most significant on the even bus. The instruction sequencer swaps the two words before using them.

It is the responsibility of software to maintain instruction cache coherency. Thus, when the virtual memory system swaps in a new page, the instruction cache entries may no longer be valid because a particular logical address may now map to a different physical address. The 88110 provides a fast (approximately five clock cycles) cache-invalidate and cache-line-invalidate feature that enables supervisor routines to eliminate stale data from the cache. Instruction cache coherence with other caches in a multiprocessor system is not normally a problem because processors do not frequently write into instruction space. In fact, the 88110 hardware does not directly support self-modifying code—that is, a program that writes into the currently executing instruction stream. However, the operating system does need to implement loaders, computed programs, copying garbage collectors, and other such programs.

Data cache. The data cache's organization resembles that of the instruction cache's. It is 8 Kbytes in size, two-way set associative, and has eight words in each line. It has a single-clock-cycle access time and can provide 64 bits each clock cycle to the load/store unit. The normal cache-write policy is "store-in" (write-back with write-allocate). And, as with the instruction cache, burst line fills begin on the missed word and data is forwarded and streamed off the bus, through the load/store unit, and directly into the register files to minimize miss latency.

We selected store-in policy because bus traffic is less for store-in caches than for store-through caches.[23] In store-through caches, the number of main memory references is never less than the store frequency regardless of cache size.[24] Considering that 20-30 percent of memory references are typically stores, this can be a problem. Store-in policy helps in multiprocessor systems, where bus utilization must be kept low for good system performance.

On a load or store that hits the cache, the memory reference accesses the cache directly. On a miss, cache control logic selects a line in the cache for replacement, using a pseudorandom selection algorithm that gives priority to invalid lines. If the line selected for replacement has not been modified since being brought into the cache, the hardware simply brings in a new cache line from memory to overwrite it. If the selected line has been modified, it is first copied back to memory before the new line comes in to replace it. A store that hits the cache on a clean line, writes directly into the cache and also broadcasts a message to other processors on the bus to invalidate any copies of this cache line they may have in their local caches.

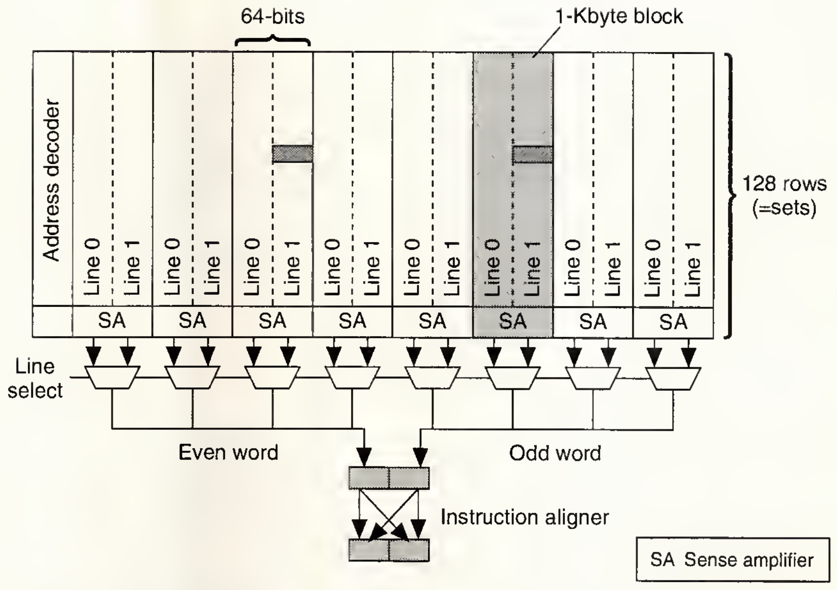


Figure 21. Cache organization.

The hardware automatically maintains data cache coherence. The data cache employs the four-state MESI cache coherency protocol illustrated in Figure 22 on the next page. A write-invalidate procedure guarantees that only one processor on the bus has a modified copy of any given cache line at the same time.

The coherency protocol is enforced by bus snooping, whereby each processor watches (snoops) all bus transactions to track the proper state for each cache line.[25] For example, if a bus transaction occurs for a cache line that a processor happens to have in the modified state, it forces the originator of the transaction off the bus, copies the modified line back to memory, changes the state of its line to shared-unmodified, and then allows the original bus transaction to be retried. The cache maintains a separate set of address and state tags for snooping, so that bus snooping does not interfere with the processor's access to its local cache.

Although hardware fully maintains data cache coherence from a multiprocessor point of view, the operating system must still flush stale data out of the cache when the virtual memory map is altered. The data cache can be invalidated quickly on a line or entire-cache basis. The cache can be cleaned (copy-back of dirty lines) or flushed (copy-back of dirty lines with invalidation) on a line, page, or entire-cache basis. The operating system activates invalidation and flushing operations by writing commands to cache control registers accessible only from supervisor mode. Invalidation operations are very fast, requiring approximately five clock cycles for either line or full-cache invalidations. Cleaning or flushing on a page or entire-cache basis requires one clock cycle for each cache set (each cache contains 128 sets) plus the memory transfer time needed to copy back any dirty lines.

External bus interface

The 88110 processor has a high instruction throughput and therefore generates a high rate of memory accesses. The on-chip caches provide relatively high hit rates and eliminate most off-chip memory accesses, but even so a substantial amount of external memory traffic can occur. To keep bus usage down to a point that a tightly coupled, dual-processor system is viable, we used a store-in (write-back) data cache policy to reduce bus traffic and also developed an efficient multiprocessor bus.

The 320-Mbyte/s, synchronous, demultiplexed, pipelined bus supports a retry cache coherency snooping protocol and offers burst-mode and split-transaction transfers. A 64-bit data path minimizes data transfer time on the bus; burst-mode cache-line fills reduce transaction overhead; a split-transaction protocol allows other masters to use the bus while another waits for memory; and address pipelining allows memory

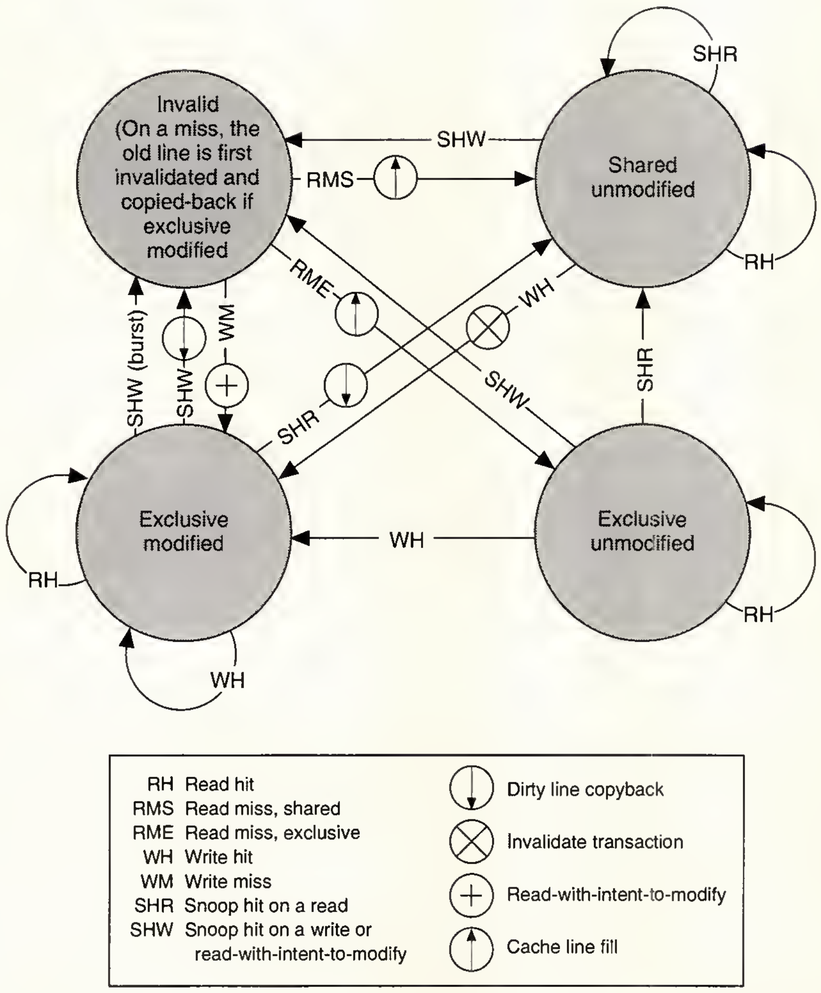


Figure 22. MESI cache coherency protocol.

access time to overlap data transfer time. These features, along with the snoopy data cache, make simple, low-cost multiprocessor systems practical.

Bus arbitration, handshaking, and data transfers are all synchronous with the system clock and are referenced to a single clock edge. An internal, analog, phase-locked loop circuit minimizes skew between the internal clock cycle and external signals referenced to the system clock. This circuit greatly simplifies the problem of electrically interfacing to the chip at high speed.

A centralized controller uses a simple bus-request/grant protocol to arbitrate bus ownership. The arbiter may "park" a processor on the bus to eliminate arbitration latency in the frequent cases that bus ownership does not change between successive transfers.

The 32-bit address bus is separate from the 64-bit data bus to support address pipelining. Address pipelining allows the address phase of a bus transaction to run concurrently with a previous data transfer phase. In multiprocessor configurations, address pipelining allows memory access times to overlap data transfer times, thereby increasing available bus bandwidth.

In the past, most microprocessor buses used a tenured transaction protocol. A tenured protocol ties up the bus from the time a transaction starts until the entire memory cycle completes and data returns to the processor. In a DRAM system with relatively long access time, this protocol wastes considerable bus bandwidth. The 88110, on the other hand, uses a split-transaction bus protocol, which allows a bus transaction to be split into distinct address and data phases that are controlled independently.

For example, a processor can send an address request to the memory system and then permit other processors to use the bus while it waits for a response. This protocol uses the bus more efficiently, consuming bandwidth only during the time addresses or data actually transfer. Address pipelining and split transactions permit the 88110 to more closely approach the theoretical bus bandwidth limit than microprocessors that use tenured bus protocols.

All burst-mode transactions transfer an entire cache line. A burst transfer uses four data beats; each beat transfers 8 bytes of data. The system controls the length of time of each data beat, which can be as short as one clock cycle.

The diagram in Figure 23 shows a possible sequence of bus transactions in a multiprocessor system (for clarity some control signals have been omitted). On the first clock cycle shown, a processor (CPU A) is parked on the bus (BG A preasserted) and requests a data cache line fill by asserting Transfer Start (TS) and driving the address bus. Two clock cycles later, the memory acknowledges receipt of the address (AACK), and CPU A terminates its address phase. Meanwhile, a second processor (CPU B) has asserted Bus Request (BG B) for an instruction cache line fill and has been scheduled next onto the bus by the arbiter's assertion of Bus Grant (DBG B) to it. As soon as CPU A relinquishes the address bus, CPU B starts its cycle. In this example, CPU B's request gets serviced immediately by memory, and the arbiter allows it to read data by granting it access to the data bus (DBG B asserted). The data transfer here is shown occurring at full bus speed; however, the memory system could use Transfer Acknowledge (TA) to pace the transfer by inserting wait states on each data beat. As soon as the transfer to CPU B finishes, the arbiter grants data bus to CPU A (DBG A) for its data transfer.

The bus-snooping mechanism, illustrated in Figure 24, enforces cache coherency among all processors on the bus. When a processor (in this example, CPU A) that is currently the bus master puts a global (GBL) address out on the bus, all other processors on the bus snoop the address. If one of these processors (CPU B in this case) has a modified copy of the data being requested in its local cache, it signals that fact to the requesting processor (CPU A) via a snoop status signal (SSTAT B). Upon seeing the snoop hit signal, CPU A aborts its bus transaction and relinquishes control. The snooping processor (CPU B) then takes control of the bus and copies its modified line back out to memory. When this transaction is complete, CPU A retries the original transaction, which now completes normally since all caches are consistent with memory. The control signals are flexible enough to support more sophisticated protocols such as intervention with direct cache-to-cache transfer (snarfing).[6]

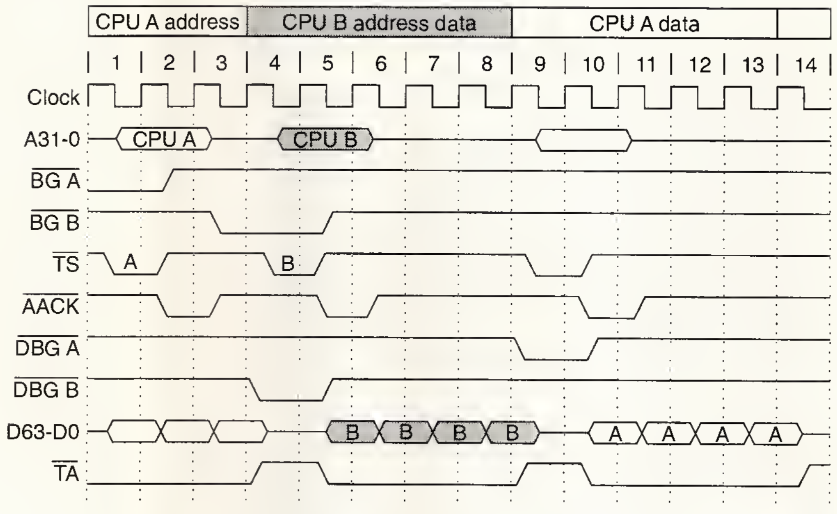


Figure 23. Split bus transaction.

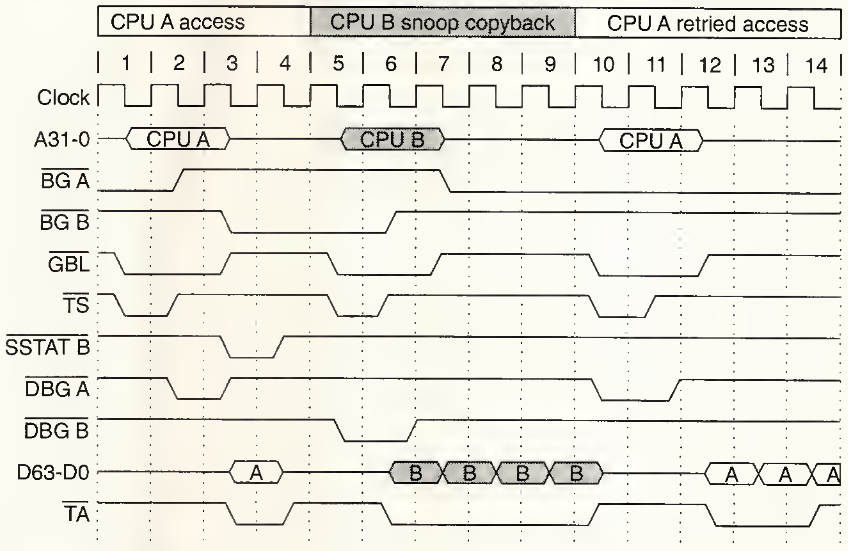


Figure 24. Retry bus-snooping protocol.

System features

The 88110 includes several features designed to improve system debugging, reliability, and testability.

One of the few drawbacks of on-chip caches is that they filter external memory references, which limits visibility and reduces the utility of in-circuit emulators for software debugging. One software debug issue, for example, is detecting the source of corruption of a particular program variable or data structure. The 88110 addresses this problem by providing two data breakpoint registers that allow a program to trap to a software debugger on an access to, or modification of, a specified logical address or range of addresses (byte, half, word, double, quad, ..., page). The 88110 also has a facility that allows a debugger to single-step a program one instruction at a time.

Two features improve the 88110's applicability in high-reliability systems: data bus parity and deterministic lockstep operation. On all external data bus write operations, the processor generates an odd parity bit for each byte transferred on the bus. On data bus read operations, the processor checks the parity bits and generates an interrupt if any byte transfers in error. For redundant systems, lockstep operation makes it possible to shadow one 88110 with another; once synchronized, two 88110s will stay in lockstep so long as they are

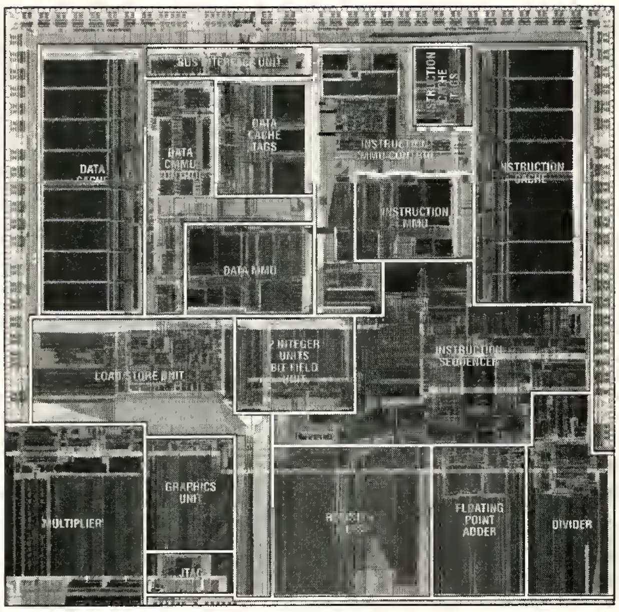


Figure 25. Photograph of 88110 die.

Table 1. Simulated SPEC ratios at 50 MHz.

Benchmark	Ratio
Gcc	46.5
Espresso	48.1
LI	57.0
Eqntott	52.9
Spice2g6	34.7
Doduc	41.4
Nasa7	67.9
Matrix300	357.8
Fpppp	64.4
Tomcatv	72.2
Geometric means	
Integer	51.0
Float	73.9
Combined	63.7

presented with the same inputs at the same time.

We improved the in-system testability of the 88110 by providing JTAG/IEEE 1149.1 boundary scan logic on all relevant I/O pins.

Silicon

We designed the 88110 in a triple-level metal, double-level polysilicon CMOS process. We used 1-μm design rules with transistor channel lengths reduced to an effective length of less than 0.8 μm. The die is easily shrinkable to 0.8-μm (0.65-μm effective channel length) technology without design modifications. The complete design required less than 1.3 million transistors and fits on a die 15 μm on a side (Figure 25). The cache SRAM cells are a four-transistor, NMOS, polyload, bit-cell design, which uses a P-well process for high immunity to soft errors.

Initially, we plan to provide the 88110 in a through-hole, ceramic pin grid array package. The 299-pin, 20 × 20, cavity-down package measures approximately 2 inches on a side, with 100-mil pin pitch.

Performance

Official benchmark data from real systems and production compilers is not available at the time of this writing. However, a good-quality prototype optimizing compiler (the Motorola 88110 Alpha complier) is available, as well as a clock-for-clock instruction simulator that accurately models all processor pipeline, primary cache, TLB, and memory system effects. Results indicate a Dhrystone 2.1 performance that translates to well over 100 VAX MIPS.

We also used the instruction simulator to run the SPEC benchmark suite at 50-MHz with a 180-ns (9/1/1/1) DRAM memory system. The results appear in Table 1. These benchmarks were compiled with the Motorola 88110 Alpha compiler, except LI, which was compiled with the Diab 88110 Compiler Version 2.37. The Nasa 7 and Matrix 300 benchmarks were preprocessed by the Kuck and Associates preprocessor. In a recent publication, Mike Phillip reports more completely on the 88110 compilers and performance.[26]

The instruction simulator does not yet accurately model all effects of external secondary-cache misses, so we haven't reported results with a second-level cache here. However, simulations with an infinite secondary cache show a combined Specmark above 80.

A significant characteristic of the 88110 is that it makes parallel instruction execution fairly easy to achieve in practice. Relatively simple compilers can produce effective code schedules for the 88110; in fact, the processor realizes substantial parallelism even on code originally generated for the 88100 single-issue CPU. The efficiency of superscalar issue ranges from 20 percent to over 50 percent, depending on the benchmark and memory configuration. Currently, over the SPEC benchmark suite, we find that two instructions issue on roughly half (ranging from about 35-70 percent) of the clock cycles on which an instruction executes at all. Of course, we expect these results to continually improve with advances in our compilers.

The increasing use of high-level languages makes good

$$[X,Y,Z,H] * \left|\begin{array}{cccc} A0 & B0 & C0 & D0 \\ A1 & B1 & C1 & D1 \\ A2 & B2 & C2 & D2 \\ A3 & B3 & C3 & D3 \end{array}\right|$$

```
Clock 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    |-|-|-|  fmul          |-|-|    st                |-|-|-|  fadd
    |-|-|    ld            |-|-|-|  fmul              |-|-|-|  fmul
      |-|-|-|  fmul          |-|-|-|  fadd              |-|-|-|  fadd
      |-|-|    ld              |-|-|-|  fadd            |-|-|-|  fmul
        |-|-|-|  fadd          |-|-|-|-| ld                |-|-|    st
        |-|-|-|  fmul            |-|-|-|-| fadd           |-|-|-|  fmul
          |-|-|-|-|-| ld         |-|      add               |-|-|-|  fadd
          |-|-|-|  fmul            |-|      add             |-|-|-|  fmul
            |-|-|-|  fadd          |-|-|    st                |-|-|-|  fadd
            |-|-|-|  fmul            |-|-|-|  fmul            |-|-|-|  fmul
              |-|-|-|  fmul          |-|      sub               |-|-|-|  fadd
              |-|      add             |-|-|-|  fmul            |-|-|-|-| fmul
                |-|-|-|  fadd          |-|-|    ld                |-|-|    st
                |-|-|-|  fmul            |-|-|-|  fmul            |-|-|-|-| fmul
                  |-|-|-|  fadd          |-|-|    ld                |-|-|-|  fadd
                  |-|-|-|  fmul            |-|-|-|  fmul            |-|      add
                    |-|-|-|  fadd          |-|-|    ld                |-|-|-|  fadd
                    |-|-|-|  fmul            |-|-|-|  fadd            |-|-|-|-|-| ld
                      |-|-|    st            |-|-|-|  fmul              |-|-|-|  fadd
                      |-|-|-|  fmul            |-|-|-|  fadd            |-|-|-|  fmul
                        |-|-|-|  fadd          |-|-|-|  fmul              |-|      add
                        |-|-|-|  fmul            |-|-|-|  fadd              |-|-|    st
                          |-|-|-|  fadd          |-|-|-|  fmul              |-|-|-|  fmul
                          |-|-|-|  fmul            |-|-|    st                |-|-|-|  fadd
                            |-|-|-|  fadd          |-|-|-|  fmul              |-|-|-|-| st
                            |-|-|-|  fmul            |-|-|-|  fadd              |-|      bcnd.n
                                                     |-|-|-|  fmul              |-|      add
```

Figure 26. Floating-point matrix multiply.

performance on general compiled code essential. But many programs spend a great deal of time in a few critical routines. System response time can dramatically improve by tuning a few of these hot spots. DSP algorithms for voice processing, graphics library routines, video processing, and interactive user interface routines are prime candidates for this type of tuning. As an example, the double-precision floating-point matrix multiply routine often used in graphics viewpoint transformations is illustrated in Figure 26. On this code the 88110 can issue two instructions on nearly every clock cycle and can sustain 97.5 MIPS and 68 double-precision Mflops (at 50 MHz), even if the point vectors being transformed are not in the cache and the processor is operating into DRAM.

Second-level cache

Although the hit rate of the 88110's internal caches is quite high, long DRAM latency and high bus utilization can still limit performance. For ultimate performance, or for system designs calling for more than two tightly coupled processors, we must further reduce memory access time and bus use.

One obvious approach is to use a secondary cache local to each processor.[27] Motorola designers developed a fully integrated second-level cache, consisting of the 88410 cache controller and an array of 62110 cache SRAMs. We implemented the secondary-cache function as a separate chip, rather than putting the logic on the 88110 itself, so that low-end systems don't have to pay for transistors they don't use.

The 88410 sits directly in the 88110 address bus path, as shown in Figure 27, and provides all secondary-cache con-

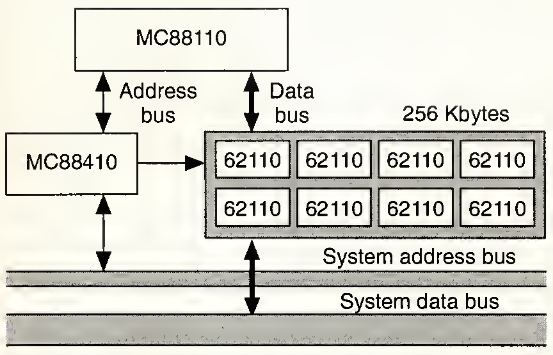


Figure 27. Second-level cache.

trol functions and tags for 256 Kbytes to 1 Mbyte of cache. Cascaded 88410s can support larger cache sizes. The cache tags included on the 88410 allow all hit, miss, and data-steering decisions to be made quickly without accessing off-chip SRAMs. This approach also reduces pin count and the number of SRAM packages required, thereby minimizing the system cost.

The SRAM cache array sits directly in the 88110 data bus path and the 88410 controls all data transfers into, out of, and through the array. The 62110 cache SRAM device works especially well in an 88110/410 secondary-cache system. The 62110, based on a standard 32K × 9, 12-ns SRAM, has a dual-bus architecture that allows data to be fed directly from the system bus onto the 88110 data bus and simultaneously captured in the internal SRAM array. We plan to offer the 62110 commercially as a cache part for other systems as well.

The secondary cache implemented by the 88410 is a direct-mapped cache with a store-in (write-back) write policy. Line length is configurable to either 32 or 64 bytes. Cache hits, using the 62110 SRAM, present a 3/1/1/1 memory cycle to the 88110.

A four-state, MESI protocol enforced by bus snooping maintains horizontal coherency between the 88410 cache and other caches on the system bus. The 88410 uses inclusion[28] to maintain vertical coherency between the 88110's internal cache and the secondary-cache array.

The bus protocol and electrical interface used by the 88410 are similar to those used by the 88110. As a result, one can design a system that can accept either an 88110 or an 88110/88410/62110 module. The 88410 also has an option to allow the system bus to operate at half the speed of the 88110 bus. This feature relaxes system timing constraints and will eventually allow systems to accommodate higher frequency 88110/410 modules, using standard TTL electrical interfaces.

IN DESIGNING THE 88110, our goal was to produce a high-performance, general-purpose microprocessor at a cost consistent with use in low-cost personal computers and workstations. We accomplished this goal with an advanced superscalar architecture and a high level of circuit integration implemented in a fine-geometry, high-yield, semiconductor fabrication process.

Acknowledgments

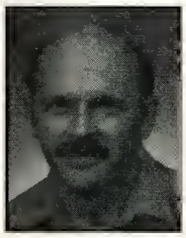
The 88110's performance and innovation resulted from the efforts of many people on the design and support teams. Bill Moyer provided technical leadership on the circuit design. Dave Mothersole, Jim Klingshirn, and Bill Moyer skillfully directed the design team. Similar acknowledgment goes to the software development team led by Anne-Marie Larkin, the 88410 design team led by Russell Stanphill, and the 62110 design team led by Brian Peterson. Special mention goes to Mike Phillip for his technical expertise in making a world-class compiler available for the 88110. Thanks also to Yoav Talgam and Mitch Alsup for their seminal work on the original 88000 architecture, which greatly simplified the task of building a high-performance superscalar microprocessor.

References

1. T. Agerwala and J. Cocke, "High-Performance Reduced Instruction Set Processors," IBM Tech. Report, N.Y., Mar. 1987.
2. G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," *Proc. AFIPS 1967 Spring Joint Comp. Conf.*, Apr. 1967, pp. 483-485.
3. J.A. Fisher, "Very Long Instruction Word Architectures and the ELI-512," *Proc. 10th Annual Int'l Symp. Computer Architecture*, June 1983, pp. 140-150.
4. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publ., San Mateo, Calif., 1990.
5. N. Jouppi and D. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," *Proc. Third Conf. Architectural Support for Programming Languages and Operating Systems*, ACM, Apr. 1989, pp. 272-282.
6. *Futurebus+, Logical Layer Specification, Draft 8.02, P896.1/D8.02*, CS Press, Los Alamitos, Calif., Sept. 1989.
7. Systems Performance Evaluation Cooperative, *SPEC Fact Sheet*, Waterside Associates, Fremont, Calif., 1989.
8. *ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic*, IEEE, Piscataway, N.J., 1985.
9. G. Radin, "The 801 Minicomputer," *Proc. Symp. Architectural Support for Programming Languages and Operating Systems,*

ACM, Mar. 1982, pp. 39-47.

10. D.A. Patterson and C.H. Sequin, "A VLSI RISC," *Computer*, Vol. 15, No. 9, Sept. 1982, pp. 9-21.
11. R.M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM J. Research & Development*, Vol. 11, No. 1, Jan. 1967, pp. 25-33.
12. J. Smith, "Dynamic Instruction Scheduling and the Astronautics ZS-1," *Computer*, Vol. 22, No. 7, July 1989, pp. 21-35.
13. S. Weiss, and J. Smith, "Instruction Issue Logic for Pipelined Supercomputers," *Proc. 11th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, 1984, pp. 110-118.
14. J.E. Thornton, *Design of a Computer—The Control Data 6600*, Scott, Foresman, and Co., Glenview, Ill., 1970.
15. D. Lilja, "Reducing the Branch Penalty in Pipelined Processors," *Computer*, Vol. 21, No. 7, July 1988, pp. 47-55.
16. D.A. Patterson and C.H. Sequin, "RISC-1: A Reduced Instruction Set VLSI Computer," *Proc. Eighth Annual Int'l Symp. Computer Architecture*, May 1981, pp. 443-457.
17. J. Smith and A. Pleszkun, "Implementing Precise Interrupts in Pipelined Processors," *IEEE Trans. Computers*, Vol C-37, No. 5, May 1988, pp. 562-573.
18. D. Kroft, "Lockup-free Instruction Fetch/Prefetch Cache Organization," *Proc. Eighth Int'l Symp. Computer Architecture*, May 1981, pp. 81-87.
19. C. Chi and H. Dietz, "Improving Cache Performance by Selective Cache Bypass," *Proc. 22nd Annual Hawaii Int'l Conf. on Systems Sciences,* Vol. 1, CS Press, 1989, pp. 277–285.
20. M. Dubois, C. Scheurich, and F. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors," *Computer*, Vol. 21, No. 2, Feb. 1988, pp. 9-21.
21. T. Kilburn et al., "One-Level Storage System," *IRE Trans. Electronic Computers*, Vol. EC-11, IEEE Service Center, Apr. 1962, pp. 223-235.
22. D.L. Black et al., "Translation Lookaside Buffer Consistency: A Software Approach," Carnegie Mellon Univ., Tech. Report CMU-CS-88-201, Pittsburgh, Dec. 1988.
23. R.L. Norton and J.L. Abraham, "Using Write-Back Cache to Improve Performance of Multiuser Multiprocessors," *Proc. Int'l Conf. Parallel Processing*, 1982, pp. 326-331.
24. J. Bell, D. Casasent, and C.G. Bell, "An Investigation of Alternative Cache Organization," *IEEE Trans. Computers*, Vol. C-23, No. 4, Apr. 1974, pp. 346-351.
25. R.H. Katz et al., "Implementing a Cache Consistency Protocol," *Proc. 12th Annual IEEE Symp. Computer Architecture*, 1985, pp. 276-283.
26. M. Phillip, "Performance Issues for the 88110 RISC Microprocessor," *Proc. Compcon*, Feb.,1992.
27. J. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. 10th Annual Int'l Symp. Computer Architecture*, June 1983, pp. 124-131.
28. W.H. Wang, J.L. Baer, and H. Levy, "Organization and Performance of a Two-Level Virtual-Real Cache Hierarchy," *Proc. 16th Annual Symp. Computer Architecture*, CS Press, 1989, pp. 140-148.



Keith Diefendorff, a member of the technical staff in Motorola's RISC Division in Austin, Texas, held responsibility for the definition of the 88110. He currently works with Apple and IBM architects to define the Power PC RISC architecture and investigates advanced microarchitecture concepts for future processors. Previously, he was a senior member of the technical staff at Texas Instruments where he designed ICs and was the primary system architect of the Explorer and micro Explorer symbolic processing LISP workstations.

Diefendorff holds an MSEE from the University of Akron. He is a member of the IEEE and the Computer Society.

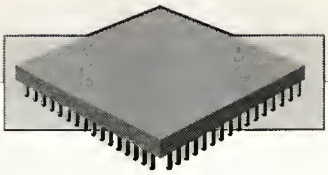
Michael Allen, a senior systems designer in the same RISC Division, helped define the 88110 bus interface. He currently works with Apple and IBM to define the bus interfaces for Power PC-based microprocessors. Before joining Motorola, he designed board-level systems and worked as an architect for the DP8344 RISC integrated microprocessor at National Semiconductor. Allen holds a BSEE from the University of Texas at Arlington.

Direct questions concerning this article to Keith Diefendorff, Motorola Inc., MDOE215, 6501 William Cannon Drive West, Austin, TX 78735; keithd@oakhill.sps.mot.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

High 159	Medium 160	Low 161



The Proposed SSBLT Standard Doubles the VME64 Transfer Rate

A revision to the IEEE 1014 VMEbus standard will offer a source synchronous block transfer protocol that doubles the transfer rate without changing the backplane or electrical interface. The faster rate in turn doubles the performance/cost ratio of the bus.

Jack Regula

Force Computers

In 1991, the IEEE P1014R (Revision D) working group drafted a new transfer mechanism for the 64-bit VMEbus:[1] the source synchronous block transfer (SSBLT). The working group gave preliminary approval to the SSBLT protocol as described here; it is thus on its way to becoming an IEEE standard.

With SSBLT, the source of the data supplies the clock used to sample the data at the destination. Consequently, the working group applied the term *source synchronous block transfer* to the protocol. SSBLT achieves higher performance by eliminating the protocol delays built into the original VMEbus specification. It is optimized by its source synchronous nature, which minimizes the skew between the data and the clock.

SSBLT doubles the rate at which data transfers between masters and slaves. Operating over the 64-bit VMEbus (as defined in the latest proposed draft, Rev. D), data transfers at 20M transfers per second times 8 bytes per transfer, for a burst transfer rate of 160 Mbytes/s.

Significantly, this performance improvement results purely from protocol improvements. SSBLT allows transfers to make use of standard VMEbus backplanes and driver technology and permits systems employing SSBLT to be backward compatible with present IEEE 1014 VMEbus modules.

Progress

VMEbus performance has increased in several steps since it was introduced 10 years ago. From the original maximum transfer rate of 10 to 20 Mbytes/s without block transfers, VMEbus throughput increased to a peak of 30-40 Mbytes/s when using 32-bit block transfers. Block transfers raise performance levels because, after an initial access latency, many slaves can supply data at a higher rate. VMEbus handshaking and protocol delays limit this rate to something less than 10M transfers per second.

Multiplexed block transfers (MBLTs) were proposed about three years ago and began reaching production status during 1991. MBLTs double block transfer performance by doubling the data path width. But, since they use the same protocol and timing rules as block transfers, MBLTs are also limited to less than 10M transfers/s.

MBLTs employ address/data multiplexing to double the data path width and, optionally, the address width. During the first cycle of an MBLT, which is conveniently called the address phase, no data transfers over the bus. In addition, another 32 bits of address are multiplexed onto the

0272-1732/92/0400-0064$03.00 © 1992 IEEE

READER SERVICE CARD

IEEE MICRO

April 1992 issue (card void after October 1992)

Name ______________________________

Title ______________________________

Company ______________________________

Address ______________________________

City ____________________ State ______ Zip ________

Country ______________ Phone(________) __________

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

READER SERVICE CARD

IEEE MICRO

April 1992 issue (card void after October 1992)

Name ______________________________

Title ______________________________

Company ______________________________

Address ______________________________

City ____________________ State ______ Zip ________

Country ______________ Phone(________) __________

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150 151 152 180 181 182
153 154 155 183 184 185
156 157 158 186 187 188
159 160 161 189 190 191
162 163 164 192 193 194
165 166 167 195 196 197
168 169 170 198 199 200
171 172 173 201 202 203
174 175 176 204 205 206
177 178 179 207 208 209

Product Information
(Circle the numbers to receive product information)

1 21 41 61 81 101 121 141
2 22 42 62 82 102 122 142
3 23 43 63 83 103 123 143
4 24 44 64 84 104 124 144
5 25 45 65 85 105 125 145
6 26 46 66 86 106 126 146
7 27 47 67 87 107 127 147
8 28 48 68 88 108 128 148
9 29 49 69 89 109 129 149
10 30 50 70 90 110 130
11 31 51 71 91 111 131
12 32 52 72 92 112 132
13 33 53 73 93 113 133
14 34 54 74 94 114 134
15 35 55 75 95 115 135
16 36 56 76 96 116 136
17 37 57 77 97 117 137
18 38 58 78 98 118 138
19 39 59 79 99 119 139
20 40 60 80 100 120 140

SUBSCRIPTION CARD

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑ **YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only $23 for a year's subscription (six issues).

Society: ______________________________

IEEE membership no: ______________________________

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate ($11.50) for three bimonthly issues.

❑ **YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other professional society, pay the sister-society rate of only $39 for a year's subscription (six issues).

Organization: ____________________ Membership no: ______________

For airmail option, see page 2.

____
Full Signature Date

____
Name

____
Street

____
City

____
State/Country ZIP/Postal Code

❑ Payment enclosed *Residents of CA, DC, Canada, and Belgium add applicable sales tax.*

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Expiration date

_ _ _ _ _ _ _ _
Month *Year*

Prices valid through 12/31/92
04/92 MICRO

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Circ., PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: ____________________

I disliked: ____________________

I would like to see: ____________________

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader service inquiries, see other side.

PO Box is for reader service cards only.

PLACE
POSTAGE
HERE

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

Editorial comments

I liked: ____________________

I disliked: ____________________

I would like to see: ____________________

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader service inquiries, see other side.

PO Box is for reader service cards only.

PLACE
POSTAGE
HERE

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
PO BOX 3014
LOS ALAMITOS CA 90720-9804
USA

data lines whenever the A64 mode (64 bits of address) is in use. After the first DTACK signal assertion (DTACK*), both the address and data lines can be used for data. From this point on, the timing for MBLTs is the same as for 32-bit block transfers. Therefore, performance doubles with MBLTs.

The 64-bit address capability added to VMEbus with MBLTs also significantly extended its useful life. And, to address the increased use of bus bridges in future systems, Revision D for SSBLTs adds a cycle retry function intended to allow deadlocks to be broken. All these enhancements are compatible with or variations of the original, asynchronous, four-edge, strobe-acknowledge VMEbus handshake.

The key to the SSBLT's ability to double transfer rates is its elimination of several protocol delays included in the original VMEbus standard.

The SSBLT mechanism goes beyond that of the MBLT by eliminating the strobe acknowledge handshake that limits the performance of the asynchronous protocol. Like the MBLT, SSBLT multiplexes address and data lines to form a 64-bit data path. The address phase is identical to that of the MBLT and can include either 32 or 64 bits of address. But in the data transfer portion, data is clocked from source to destination without cycle-by-cycle handshaking at a rate of up to 20 MHz or 160 Mbytes/s.

SSBLTs contain unique address modifier codes: 07 indicates an A32 SSBLT, and 06 indicates an A64 SSBLT. Boards not capable of performing SSBLTs don't respond to these address modifiers nor assert bus error signal BERR*. Thus the master can repeat the access with another transfer method such as a standard block transfer or an MBLT. This level of interoperability is assured by requiring an SSBLT board to support all earlier transfer methods.

Transfer and throughput rates

Because the cycle-by-cycle handshake has been eliminated, boards can relatively easily transfer data at the peak transfer rate. Contrast this with VMEbus asynchronous handshaking, which is hard pressed to approach 10 MHz and is slowed down by backplane and driver propagation delays. The initial access latency amortized over the entire burst primarily determines data throughput for an SSBLT.

I've estimated that, with back-to-back transfers of 64 bytes, the sustained transfer rate using SSBLT is 128 Mbytes/s for writes and 100 Mbytes/s for reads. Increasing the block size to 2 Kbytes boosts the estimated sustained rate to 159 and 157 Mbytes/s for writes and reads.

The sustained-rate calculations assume 100 ns for the address phase on a write transfer and 240 ns for reads, including initial access latency (typical of high-performance VMEbus interfaces). Because 8 bytes transfer every cycle, each 64-byte block requires eight transfers. At the SSBLT maximum rate, transfers execute every 50 ns. Therefore, the calculations for back-to-back 64-byte blocks are

Write transfers
$$100 \text{ ns} + 8 \text{ transfers} \times 50 \text{ ns} = 500 \text{ ns/block}$$
$$64 \text{ bytes}/500 \text{ ns} = 128 \text{ Mbytes/s}$$
Read transfers
$$240 \text{ ns} + 8 \text{ transfers} \times 50 \text{ ns} = 640 \text{ ns/block}$$
$$64 \text{ bytes}/640 \text{ ns} = 100 \text{ Mbytes/s}$$

When the block size increases to 2 Kbytes, requiring 256 transfers to complete, the overhead of the address phase is amortized over a larger data transfer period. Thus, back-to-back transfers of the larger blocks yield

Write transfers
$$100 \text{ ns} + 256 \text{ transfers} \times 50 \text{ ns} = 12{,}900 \text{ ns/block}$$
$$2 \text{ Kbytes}/12{,}900 \text{ ns} = 159 \text{ Mbytes/s}$$
Read transfers
$$240 \text{ ns} + 256 \text{ transfers} \times 50 \text{ ns} = 13{,}040 \text{ ns/block}$$
$$2 \text{ Kbytes}/13{,}040 \text{ ns} = 157 \text{ Mbytes/s}$$

The write latency in these calculations assumes that the packet can be received without delay at the beginning of the transfer's data cycle. The calculations also assume that, for reads of small blocks, a FIFO queue buffers the transfers and is partially loaded before the start of a transfer. We estimate this step to require 240 ns. For large block transfers, the calculations assume the circuitry of the boards involved is fast enough to handle the transfer in either direction without throttling.

Eliminating protocol delays

The key to the SSBLT's ability to double transfer rates is its elimination of several protocol delays included in the original VMEbus standard. These delays simplified implementations by allowing architecturally simple interfaces to be implemented with logic that is both agonizingly slow and extremely modest in complexity by today's standards. Today, architectural elegance is affordable, as is subnanosecond logic.

Using the high-speed, high-density ASIC technologies available now, single-chip interfaces—including FIFO buffers and

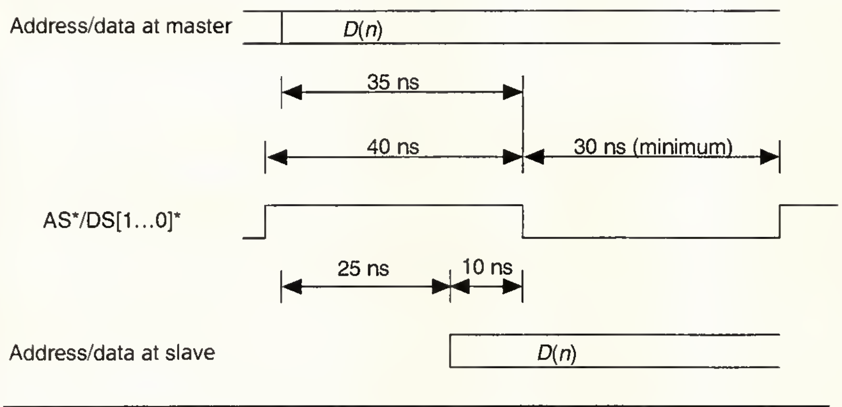


Figure 1. VMEbus protocol delays.

DMA controllers employing a 20-MHz SSBLT protocol—are well within the state of the art. Bus interface ASICs need only a few hundred additional gates to add SSBLT capability to a design that already includes MBLTs.

Protocol design for TTL backplanes is complicated by the considerations of incident wave switching. Incident wave switching[2] in a transmission line environment refers to a driver's ability to drive its output voltage across the switching threshold of receivers placed along the line as the voltage wavefront first propagates down the transmission line. A driver's incident wave output voltage is reduced by voltage division of its output impedance with the transmission line's impedance. When the driver isn't strong enough for incident wave switching, the switching threshold isn't crossed until a reinforcing reflection arrives from the far end of the transmission line. The resulting waveform then takes on a stairstep appearance with one step per reflection. In VMEbuses, a single step often appears near the threshold region.

TTL backplanes generally do not provide incident wave switching unless they are only lightly loaded. Protocol designers must take into account the possibility that certain signals, such as data strobes, might be received with incident wave switching, while transitions of the data itself might not be seen until a reflection arrives from the far end of the backplane. The original VMEbus standard provided for this situation by requiring the master to provide a 35-ns setup time while guaranteeing the slave only 10 ns of setup. The difference is two backplane delays totaling 15 ns plus an additional allowance of 10 ns for skew in the bus drivers and receivers. The two backplane delays allow time for the reinforcing reflection to arrive from the far end(s) of the backplane. The SSBLT protocol's data capture delay parameter permits the same effects. At 37.5 ns it is actually slightly more conservative than the original VMEbus protocol! Figure 1 illustrates the VMEbus protocol delays.

VMEbus has two additional protocol delays. The slave may not assert DTACK* until at least 30 ns has elapsed since assertion of DS[1..0]*. Although not shown in Figure 1, the master cannot capture read data from the bus until 25 ns after the assertion of DTACK* because of possible skew between data and DTACK*. These protocol delays mean that even with infinite speed logic and zero-delay backplanes, a compliant VMEbus data transfer cycle takes a minimum of 70 ns for writes

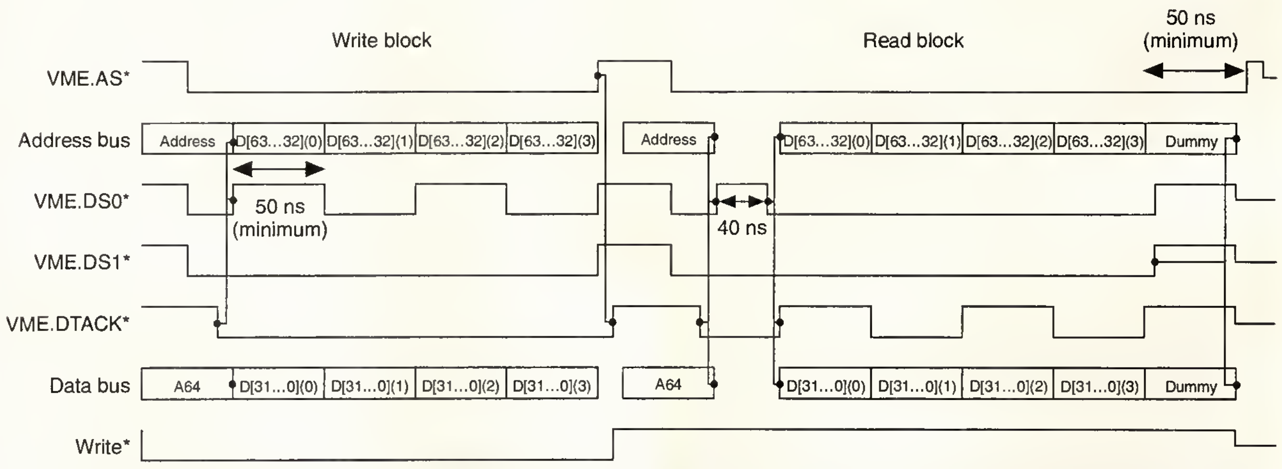


Figure 2. VME64 source-synchronous block transfer. If AS* rises before data capture time, data does not transfer and the cycle ends. The slave can sample AS* at what would have been the data capture time and verify the burst end. DS1* stays asserted throughout the burst to keep the bus timer enabled.

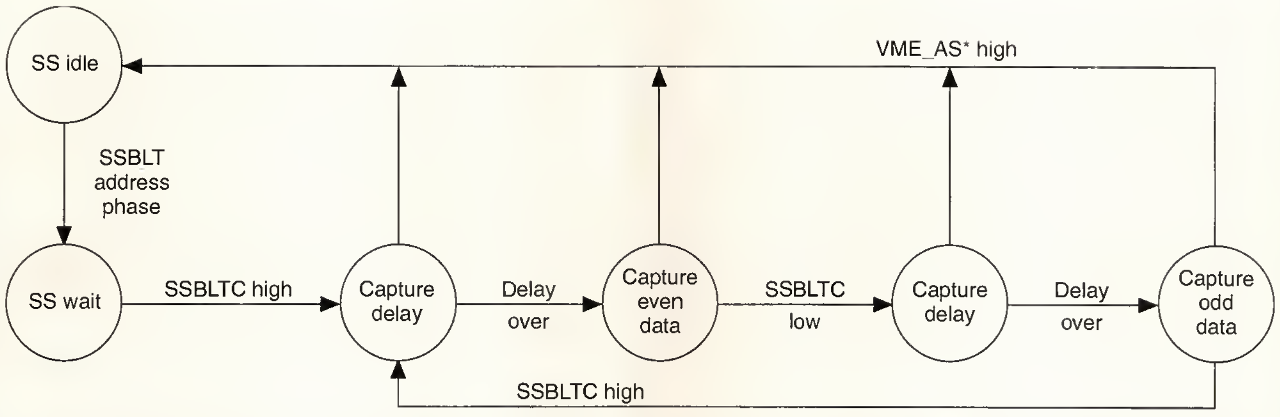


Figure 3. Asynchronous state machine for SSBLT master reads and slave writes. SSBLTC indicates master_read and DTACK* or slave_write and DS0*.

and 95 ns for reads. When practical logic, driver, receiver, and backplane delays are added to these protocol minimums, VMEbus users find it extremely difficult to achieve burst transfer rates of greater than 64 Mbytes/s, even with the MBLT method. The significance of the SSBLT method is that it makes it relatively easy to achieve burst rates of 160 Mbytes/s and to sustain throughputs of over 100 Mbytes/s.

Figure 2 shows both read and write SSBLTs. These contain an address phase in which the slave asserts DTACK* as soon as it recognizes the address and address modifiers and is ready to transfer data. In the write cycle, the master then uses DS0* to clock the data to the slave at a 20-MHz rate (or slower, if desired). In the read transfer, an additional data strobe pulse provides buffer turnaround time. Then the slave, which is the source on a read cycle, uses DTACK* to clock the data to the master.

Figure 3 shows a small asynchronous state machine that may be used by an SSBLT master to receive data on a read cycle or by an SSBLT slave to receive data on a write cycle. Note that in Figure 2 each edge of DS0* or DTACK* transfers data. The SSBLT protocol specifies that at least 50 ns must occur between each edge. The data destination detects each edge, delays a data capture time, then latches the data from the VMEbus. Data is nominally in phase with the clock at the source (±5 ns). The data capture delay, which must be between 37.5 and 45 ns, allows for nonincident wave switching, 5 ns of skew at each source, and destination and settling times.

Figure 4 illustrates SSBLT protocol delays. This protocol simply sets the instantaneous transfer rate to the maximum that can be supported (with a margin for safety) and provides the timing rules for data transfer. In contrast with a VMEbus asynchronous handshake, it does not include cycle-by-cycle handshaking delays. Such delays make performance depend upon the physical length of the backplane and the speed of the backplane drivers and interface logic. SSBLT masters and slaves need only keep their skew and capture delay errors within budget to be able to use the maximum transfer rate.

Rescinding DTACK* driver

A rescinding three-state driver is a circuit that is actively driven high and then tristated (changes voltage to high, low, and off states). When used for DTACK*, a rescinding driver speeds up asynchronous block transfers. In a heavily loaded VMEbus backplane, the time constant of the terminators and the distributed capacitance of the bus increases the propaga-

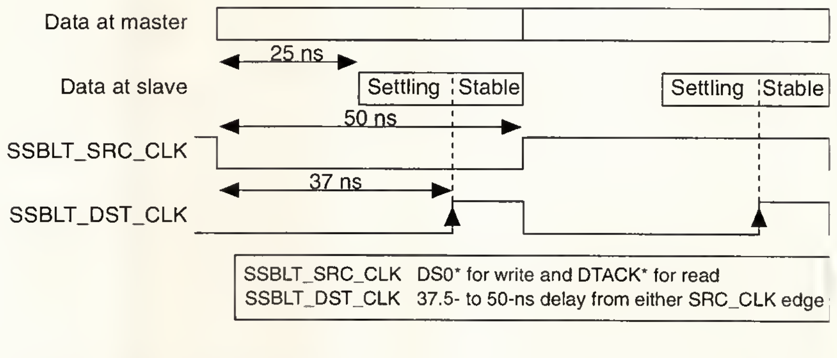

Data at master
25 ns
Data at slave
Settling
Stable
Settling
Stable
50 ns
SSBLT_SRC_CLK
37 ns
SSBLT_DST_CLK
SSBLT_SRC_CLK DS0* for write and DTACK* for read
SSBLT_DST_CLK 37.5- to 50-ns delay from either SRC_CLK edge

Figure 4. SSBLT protocol delays.

tion delay for the rising edge of DTACK*. The resulting delay is greater than the VMEbus 40-ns minimum strobe, high-time specification. This delays the start of the next cycle.

The SSBLT revision to the VMEbus standard provides the timing rules for use of a rescinding DTACK* driver. If a slave is using a tristate driver for DTACK*, it can enable its driver upon selection. It may not drive DTACK* low until 30 ns after DS[1..0] assertion and *must* drive DTACK* high within 25 ns of all strobes high (AS*, DS0*, and DS1* = 1). DTACK* must be tristated within 50 ns of all strobes becoming high—20 ns before the next selected slave is permitted to drive DTACK* low.

The SSBLT revision to the VMEbus standard provides the timing rules for use of a rescinding DTACK* driver.

VMEbus protocol does not allow multiple slave transactions that might result in slaves with three-state DTACK* drivers attempting to drive DTACK* to opposite levels. The timing rules provide a period of time in which both the previous and newly selected slaves may drive DTACK* high; however, this is not a problem. The only possibility for compatibility problems due to use of a three-state driver for DTACK* exists when the slave is participating in a proprietary broadcast scheme. In such a case, the slave's DTACK* driver can and should be controlled so as to emulate an open-collector output.

The VMEbus standard already specifies a high-current, three-state driver for DS0*; SSBLT adds that requirement for DTACK* since DTACK* must function like DS0* for SSBLT read cycles. Standard 48-mA drivers support data lines, while 64-mA drivers support DS0*, DTACK*, and other control signals.

Transfer length, burst termination

The SSBLT transfer mechanism permits from one to 256 transfers in one block, based upon the requirement that the burst ends at the first 2-Kbyte boundary. The burst can continue only after another address phase, which appears on the VMEbus as a second SSBLT. This arrangement limits the size of the address counter required at the slave and means that boards that are not involved in a transfer don't have to increment their address counters during it.

The master terminates an SSBLT by driving AS* high. For both reads and writes, if AS* changes to high before data capture time, data cannot transfer, and the cycle ends. By sampling AS* at what would have been the data capture time, the slave can determine that the burst has ended. If AS* is high, the cycle ends, and no data transfer becomes associated with the previous strobe edge. To keep the bus timer enabled and thus prevent specious error indications, DS1* is asserted throughout a burst.

Throttling

The SSBLT provides interblock throttling as a packet-level mechanism corresponding to cycle-by-cycle handshaking. Ideally, an SSBLT slave asserts DTACK* during an address phase; it signals its ability to accept/provide a burst at the full transfer rate. Subsequently, it needs to throttle only infrequently and momentarily. An intrablock throttling mechanism answers this need.

Some applications, such as digital imaging systems, involve large block transfers. It can be necessary to suspend these momentarily to allow a competing transfer to take place on the local bus of the master or slave. This is an example of an appropriate use of *intrablock* throttling.

To delay another block transfer, the destination (slave) can make use of two options. During the address phase, it can simply fail to respond until it is ready, or it can assert both RETRY* and BERR*. The source (master) then terminates the cycle before any data transfers, releases the bus, and waits before attempting another transfer. This is *interblock* throttling. Note that the RETRY* protocol specifies a bus release that usually results in other VMEbus traffic before the retry takes place.

SSBLT also provides a method for intrablock throttling. This alternative activates when the destination's input buffer is almost full, yet the burst is not over. Intrablock throttling allows the destination to suspend the data transfer until it can catch up. System designers should arrange that intrablock throttling is required only infrequently.

To employ intrablock throttling during a write, the slave drives DTACK* to a high level. When the master detects DTACK* as high, it simply suspends the transfer until DTACK* becomes low again. Figure 5 shows intrablock throttling for the slowest case in which the master doesn't suspend transfers until it has driven the third data and strobe edges after DTACK* deassertion. A faster responding source might also have paused with either D[63..0](4) or D[63..0](5) valid on the bus; destination devices must be able to deal with any of these possibilities.

During a read, the master temporarily stops the slave from transmitting data by driving DS0* high. When the master drives DS0* low again, the slave continues the transfer. Figure 6 shows the waveforms for intrablock throttling on a read. The slave's response to the deassertion of DS0* on a read is analogous to the master's response to DTACK* on a write. The same possible stopping points of one, two, or three transfers past strobe deassertion exist as in the write case.

In intrablock throttling for both reads and writes, the source

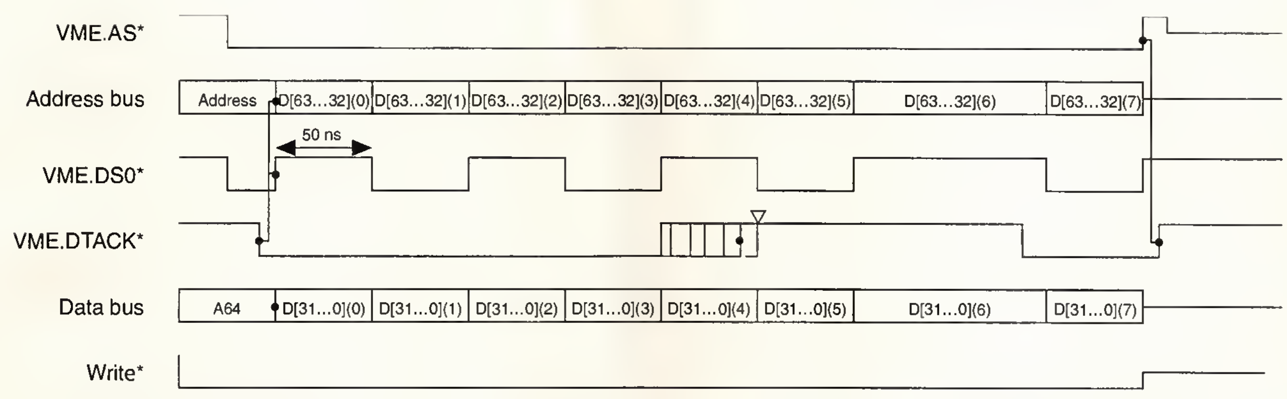


Figure 5. Write intrablock throttling. Throttling protocol rule: Source must freeze its data and clock output within 100 ns of detecting DTACK* high. Since a 30-ns round-trip path delay may exist between source and destination, the destination should throttle when its queue is within 3 locations of the overflow point.

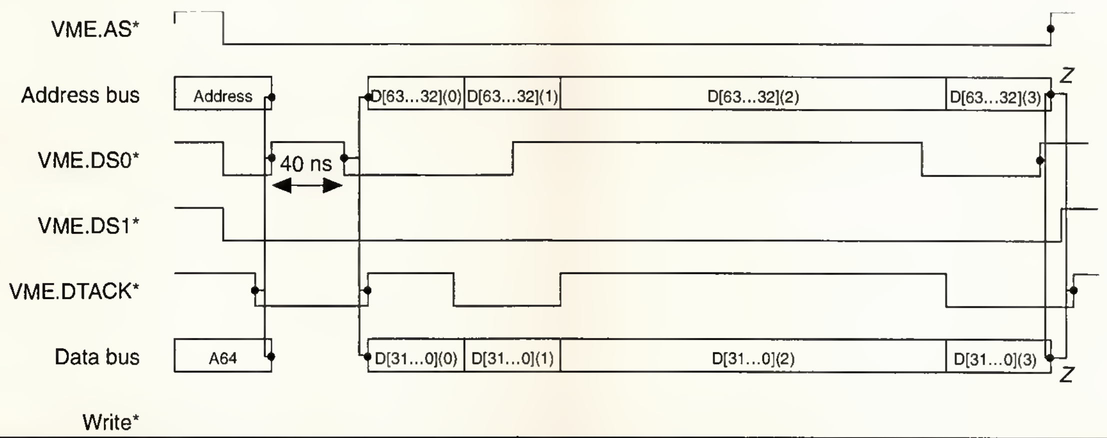


Figure 6. Read intrablock throttling.

must freeze its data and clock outputs within 100 ns of detecting the throttle signal. Note, though, that the timing for the deassertion of DTACK* or DS0* is not fixed; no specific time reference drives either signal. Similarly, no specific time reference determines when the sender of the data must sample DTACK* or DS0*. The SSBLT specification only requires the data sender to halt data transfers within 100 ns of detecting either signal as high.

In Figure 7 (on the next page) two unspecified timings relate to throttling and indicate the added timing consequences of backplane delay. The timing values in this figure also apply to error handling (more on this later).

After asserting the intrablock throttling signal, the destination must be able to accept as many as two additional transfers before the source stops transferring data. Because the round-trip backplane delay between the source and destination might be as great as 30 ns, the destination should throttle bursts when its queue is within three transfers of the overflow point.

Throttling can be misused as a slow form of cycle-by-cycle handshaking. VMEbus Rev. D will recommend that interface designs not only use throttling infrequently but also only for short periods. The spirit of the revision's SSBLT protocol is that data be burst over the bus at 20 MHz with suspensions only for infrequent, exceptional conditions such as needed for a DRAM refresh.

Conservative timing

The timing for SSBLT is based upon the same backplane and driver characteristics as the original VMEbus specification. This standard provided reliable data transfer with up to 25 ns of skew between data and the data strobe or DTACK*. This skew time includes two 7.5-ns backplane propagation delays plus 10 ns of driver/receiver skew. The two back-

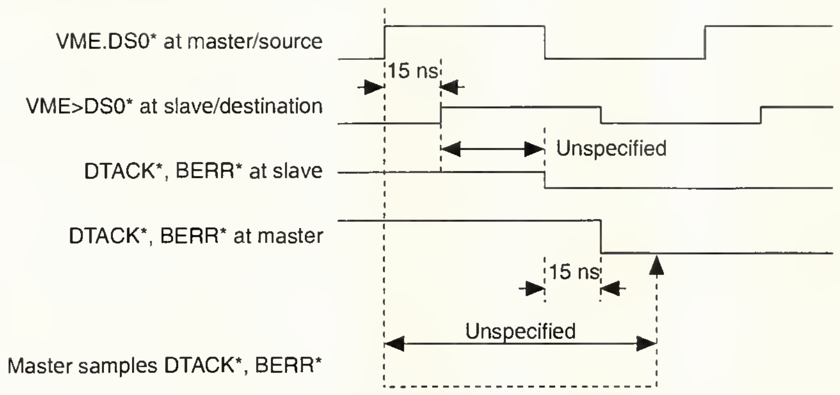

VME.DS0* at master/source
15 ns
VME>DS0* at slave/destination
Unspecified
DTACK*, BERR* at slave
DTACK*, BERR* at master
15 ns
Unspecified
Master samples DTACK*, BERR*

Figure 7. Error and throttling timing. The master must terminate or suspend transfers within 100 ns of detecting a low on BERR* or a high on DTACK*. The points at which the slave asserts BERR* or deasserts DTACK*, or at which the master samples them, are not specified. Note that RETRY* assertion is permitted only on the address phase.

Table 1. Transmission timings.

Characteristic	Timing (ns)
Data clock skew (two backplane propagation delays)	15.0
Data settling time	10.0
Driver/receiver skew	10.0
Receiver setup time	2.5
Minimum capture delay	37.5
Time quantization error (half period of ASIC clock)	5.0
Minimum hold time	2.5
Minimum transmit period	45.0

plane propagation delays appear in the potential skew because the control signal edges (which are driven with 64-mA drivers) might be seen with incident wave switching. The data edges will generally be detected only after their first reflection from the far end of the backplane.

The worst-case minimums for source synchronous capture delay and the overall transmission period take into account the same skew, settling times, setup/hold times, plus a time quantization error allowing this delay to be generated synchronously. The transmission period determines the minimum time that can be allowed between data strobe edges. Table 1 lists these times, in nanoseconds.

To provide an extra margin, the SSBLT protocol adds an extra 5 ns to the minimum transmission period for a specified transmission period of 50 ns. See Figure 4 again for the SSBLT protocol delays and stable data window.

AM codes

SSBLT employs two new address modifier codes that were previously undefined in the original VMEbus standard. They are 0x06 for A64 SSBLT and 0x07 for A32 SSBLT.

Terminating a transfer

Masters can terminate transfers when the required data has been sent or received. During a write, when the master has transmitted the required data, the master stops strobing the DS0* line and drives AS*, DS1*, and DS0* high. The slave responds by driving DTACK* high and thus terminating the transfer.

In a read, when the master does not need more data, it terminates the burst by driving DS0*, DS1*, and AS* high. The slave then terminates the transfer within two strobe edges. Figure 2 (shown earlier) illustrates normal terminations for both the read and write cycles. The delays between master and slave result in a "dummy" data cycle being driven onto the bus by the slave after the master terminates the read. The master keeps its AS* signal asserted past the end of the dummy cycle to avoid driver conflicts with the next master.

Either masters or slaves can terminate transfers after detecting an error. If a slave detects an error of any kind during a write, it terminates the transfer by asserting BERR* low. The master then terminates the transfer within two strobe edges. Figure 8 displays a write-error termination.

If a master detects an error of any kind during a read, it terminates the burst by driving DS0*, DS1*, and AS* high. If the slave detects any errors, it also terminates the read by ceasing to toggle DTACK* and asserting BERR* low. In the latter case, the master aborts the transfer. Figure 9 illustrates read-error termination.

VMEBUS WAS ORIGINALLY CONCEIVED as a combination processor-memory-I/O bus.[3] Since its inception, processor speeds have increased by a factor of over 100, forcing architects to remove most CPU-memory traffic from the bus. Despite this, the demand for higher bus speeds continues to increase because of the need to support interprocessor com-

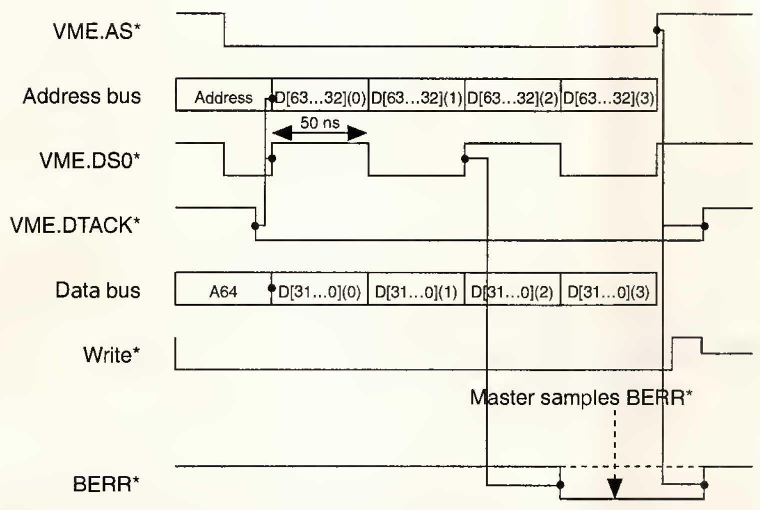


Figure 8. Write with BERR* termination.

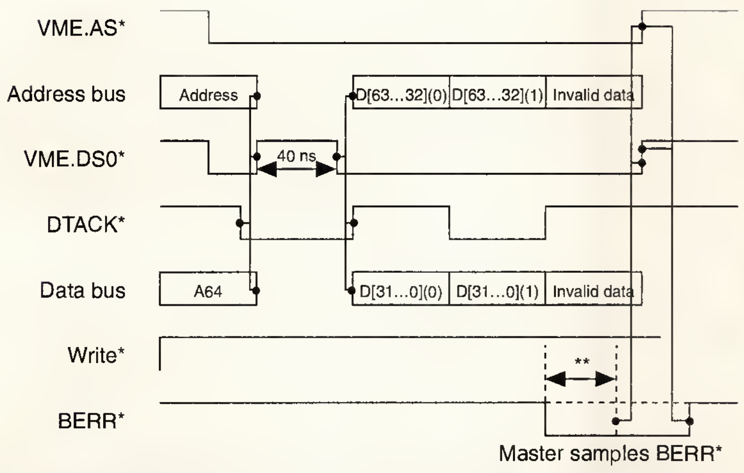


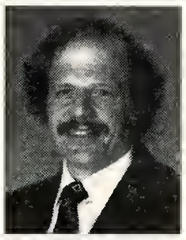
Figure 9. Read with BERR* termination. ** indicates unspecified timing.

munications at increasing rates and to support higher performance I/O for imaging and graphics, mass storage interfaces, and network communications.

The required bus performance is not a simple function of processor speed. Rather, it is strongly dependent on system architecture and application. The decision to use a particular processor and a particular backplane bus for a particular application incorporates many components other than bus performance. Preeminent among these are cost/performance and risk management. By doubling performance with only a marginal increase in cost, SSBLT doubles the performance/cost ratio of the VMEbus, widening its market and extending its life quite significantly. By adding performance headroom to the dominant 32-bit and now 64-bit backplane bus standard, SSBLT provides increased assurance that the VMEbus will continue to meet the needs of its users.

References

1. *IEEE/ANSI Standard 1014, Versatile Backplane Bus: VMEbus,* IEEE Service Center, Piscataway, N.J., 1987.
2. R.V. Balakrishnan, "The Proposed IEEE 896 Futurebus—A Solution to the Bus Driving Problem," *IEEE Micro*, Vol. 4, No. 4, Aug. 1984, pp. 23-27.
3. D.B. Gustavson, "Computer Buses—A Tutorial," *IEEE Micro*, Vol. 4, No. 4, Aug. 1984, pp. 7-22.



Jack Regula is manager of hardware engineering at Force Computers in Campbell, California. Earlier, he cofounded and served as vice president of research and development of the VME company, Ironics, Inc. He also served as technical chair of the VTC Consortium that designed and developed the VIC chip (VMEbus interface chip). He holds BSEE and MSEE degrees in electronics engineering from Rensselaer Polytechnic Institute in New York.

Direct questions about this article to the author at Force Computers, Inc., 3165 Winchester Blvd., Campbell, CA 95008-6557.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162	Medium 163	High 164

Micro Standards

PCMCIA: The other interface

Carl Warren

McDonnell Douglas

Space Systems Company

(714) 896-33ll

x. 7-1230

warren@ssdunx.mdc.com

Recently, Dante Del Corso, our editor-in-chief, sent me a note suggesting that there was a growing interest in PCMCIA, a 68-pin interface for memory cards used in notebook computers. (PCMCIA is the Personal Computer Memory Card International Association.) Dante felt that, although it isn't currently covered by any IEEE or ANSI organization, the de facto standard is having a major impact on the industry.

Agreeing with Dante, is I. Dal Allan, principal consultant at ENDL (Saratoga, California) and a recognized industry expert on interfaces. Allan concedes that the interface has grown from a convenient method of adding slim (3.3-mm thick) memory cards to notebook computers.

Interest in the 68-pin interface seems to be growing simply due to the critical mass of vendors developing notebook and laptop computers. But memory cards aren't the only thing PCMCIA supports. Designers are preparing to use the interface for disk drives as well. Moreover, with emerging 1.8-in. winchester disk drives designers see a good market for add-ins for the portable computers. The emerging interface promises to provide superior interchangeability and better reliability over the long haul. It is rated at more than 10,000 insertions and ensures compatibility over multiple vendors.

PCMCIA defines three types of interfaces. Type I defines the interface for the 3.3-mm memory card.

Type II allows the specification to accommodate storage devices that can't fit the 3.3-mm height constraint. Though the typical height is 5 mm, Type II maintains compatibility to the base standard by using a 3-mm-wide rail along the edges and a 10-mm-deep mating area, both of which are kept at the standard 3.3 mm. The upshot is that designers won't have to rework slots or cases to manage the larger card.

Still in the proposal stage is Type III, which is supposed to define the interconnection scheme for LANs (local area networks) and modem cards. This definition describes a 50-mm body extension and an 11-mm height. This cavity size seems large enough for the 1.8-in. disk drive form factors. The driving factor is the height since manufacturers want to stay within the 0.5-in. thickness for notebooks. Palmtop computers, though, may pose a different set of problems.

If you are looking for a quick solution and availability for PCMCIA extended products, don't hold your breath. Members of the committee are still wrestling with sizing. For example, three Type I cards can't fit into the Type III cavity, and that is some concern. Additionally, pin size and orientation haven't been worked out.

Among the other issues facing the PCMCIA specification writers are the number of insertions. Although the specification claims more that 10,000 insertions, it is unclear what the real number is. It may be necessary to devise a new seating and release system to minimize wear and ensure proper electrical contacts. Furthermore, PCMCIA has problems with the disk storage capacity for small drives. Consequently, some people talk about providing compression as part of the basic input/output system (BIOS). More than likely, PCMCIA vendors of storage devices will provide a fully integrated card including drive and BIOS with compression capability built in. No doubt Microsoft Corp. (Bellevue, Washington) will want to get its two cents in with a ROM version of its DOS (disk operating system). Whether the Type III cavity can accommodate a full-featured card remains to be seen.

Industry observers such as ENDL's Allan and

0272-1732/92/0400-0072$03.00 © 1992 IEEE

Richard Steincross from RMS Labs (Long Beach, California) wonder about the cost when compared to other alternatives. Allan points out that the X3T9 group has discussed porting SCSI (the small computer systems interface) to the PCMCIA world but to date has found no takers. The solution seems valid, and protocol chips exist that would support virtually any peripheral.

Even with lack of support from the SCSI community for PCMCIA, Milpitas, California-based Adaptec Inc. is considering a version of its 8000-series integrated disk controller. "That may help on the cost angle," suggests Steincross. He, however, expressed surprise that the emphasis isn't on the IDE (integrated drive electronics) specification. Steincross explains that IDE caught on quickly because it "... was cheap, easy to integrate and heavily supported by the industry. I don't see PCMCIA enjoying as much interest."

Though the jury may not be completely in on PCMCIA, proponents point out the industry infrastructure is growing. Getting on the PCMCIA bandwagon isn't necessarily inexpensive however. Executive and associate memberships carry fees of $10,000 and $2,500 a year. An executive membership buys you nine board seats, while an associate is allowed five seats. You can sign up as an affiliate, which allows you to attend meetings, observe, and receive documentation but not participate in discussions. If you are interested in membership or obtaining the latest document, call (408) 720-0107.

Reader Interest Survey
Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 180 Medium 181 High 182

Game Genie: copyrights and add-ons

Richard H. Stern

Oblon, Spivak, McClelland, Maier & Neustadt, P.C.

1755 Jefferson Davis Highway, Suite 400

Arlington, VA 22202

Users often modify computer programs to enhance their utility. Sometimes they buy add-on programs to increase performance. These add-ons provide interfaces that are easier to learn and remember, increase speed, add new functions, perform additional tasks, and otherwise interact with a preexisting program to provide additional utility. Typically, add-on programs do not permanently modify the underlying program, do not make a tangible copy of a new, modified program, and cannot be used unless the customer has already purchased a copy of the underlying program

Often, owners of copyrights in underlying programs do not object to add-on programs. The add-ons add to the utility of, and increase the demand for, the underlying programs. There are many circumstances, however, in which copyright owners might be displeased with—and therefore want to suppress—an add-on program.

Consider the case of a low-power version of a program sold cheaply for the low-price market and a high-power version sold upmarket. What if an add-on cheaply converts the down-market model to perform the tasks of the upmarket model? (See the box on a similar case.) Sometimes, an add-on program shows users the inadequacies of the underlying program by providing improvements that the copyright owner has refused to be bothered to make. That may pave the way for customers to migrate to another product. This appears to have occurred in the case of database management add-on programs, which were eventually followed by competing programs that included the features of the add-on programs, as well as still additional improvements.

I am not aware of any litigation over add-on programs of the types just described. However, a recent decision from the San Francisco federal trial court comes close. In *Lewis Galoob Toys, Inc.* v. *Nintendo of America, Inc.*,[1] the court rejected a claim that copyright owners have the exclusive right to determine whether end users may temporarily modify computer program-related works once they are in the users' hands. The underlying work in this case was a video game operated by a computer program. But the same legal conclusions would appear to apply with equal force to any other computer program-related work, such as a spreadsheet, database, or word processing program, in a consumer user's hands.

Background. Nintendo, a major Japanese and American seller of home and arcade video game equipment, owns copyrights in many popular video games. It markets these games in the home video field by selling cartridges that connect into its Nintendo Entertainment System (NES) game consoles. These are small special-purpose microcomputers that provide a video display on a home television set. The cartridges fit into the consoles as cassettes fit into audio tape and videotape players.

A typical video game, such as *Super Mario Brothers*, features a protagonist (Mario) whom the player moves across the display screen. The computer system displays obstacles and enemies which Mario must overcome. The player pushes buttons and manipulates controls to cause Mario to jump over obstacles, evade dangers, kill enemies, and traverse a series of "worlds," at the end of which he may rescue a princess from an ogre. Programmed-in constraints limit Mario's abilities. He can jump only so high or so far. He has only so many missiles to hurl at enemies. His speed is limited. To the extent that the player's abilities are insufficient, given the programmed-in constraints, to overcome the dangers that Mario

0272-1732/92/0400-0074$03.00 © 1992 IEEE

faces, he succumbs and dies. After a set number of deaths, the player loses and the game ends.

Galoob markets an accessory, the Game Genie Video Game Enhancer. Game Genie fits between a video game cartridge and the NES console. It modifies electronic signals passing between the console and cartridge by temporarily inserting code segments, or *patches*, into the computer program as it appears in temporary memory (RAM). Game Genie thus allows a user to change the constraints, for example, to make Mario jump higher, run faster, and hurl more missiles at adversaries. It can also allow Mario more deaths before the player loses. Game Genie similarly modifies the play of other NES-compatible video games.

Nintendo contended that Galoob was causing its customers to create derivative-work versions of the video game, in violation of Nintendo's copyright. Section 106(2) of the Copyright Act gives a copyright owner the exclusive right to prepare derivative works based on a copyrighted work. Section 101 of the Copyright Act defines a derivative work as a work based on a preexisting work, such as translation, dramatization, motion picture version, art reproduction, condensation, "or any other form in which a work can be recast, transfonned, or adapted."

Nintendo claimed that by modifying the program to change the rules of the game Galoob was making a change in Nintendo's copyrighted work that amounted to preparation of a derivative work. The copyrighted works in a video game, according to Copyright Office practice, include a computer program (literary work) and a visual display (audiovisual work). Since Game Genie changed the program by putting patches into the code, and since that changed the visual display, Galoob caused users to make unauthorized, derivative-work computer programs and displays.

Nintendo also markets (or licenses others to market) devices that modify its video games in various ways, such as speeding up part of a game or skipping stages. But Nintendo maintained that its copyright gives it the exclusive right to traffic in such modifications. Since Galoob caused its customers to trespass on Nintendo's claimed exclusive right to modify the game play, Nintendo accused Galoob of *contributory infringement* – meaning, contributing to, or causing customers to engage in, copyright infringement.

Galoob denied that the modifications created a derivative work. It also asserted the affirmative defense that personal game modification by end users for their personal enjoyment of games they had purchased was a fair use and was therefore privileged. The trial court agreed with Galoob on both counts, and an appeal is pending.

Is there a derivative work? Game Genie does not make a physical copy of the computer code stored in a Nintendo cartridge. Its electronics and code patches merely interact with those of the NES console and the game cartridge, modifying signals to change the video display and the results of game action.

For example, a user might set Game Genie to continue the game until Mario dies six times, rather than three. In effect, Game Genie substitutes its instructions and data for parts of the original program. (For example, "do so-and-so for $i = 1$ to 3" becomes "do so-and-so for $i = 1$ to 6.")

But this change occurs only temporarily, without rewriting the ROM in the Nintendo cartridge. Game Genie is like Maxwell's Demon. It sits between the copyrighted computer program in the cartridge and the NES console's CPU and censors the messages that go back and forth. Since it does not write anything down in a fixed form, it does not make a tangible, more-than-transitory copy of any of Nintendo's computer program or visual display. The modified code exists only in RAM, and the modified display appears only temporarily on the TV screen.

However, section 106(2) of the Copyright Act does not require that one make a permanent copy. It gives copyright owners an exclusive right to prepare derivative works in tangible or intangible form. Thus a stage performance of the musical *Cats* without authorization from T.S. Eliot would infringe the copyright in his book of poems, *Old Possum's Book of Practical Cats*, even if no written script was reproduced. Therefore, Game Genie's program modifications apparently prepare a derivative work, in terms of both the program and the audiovisual work whose display the program causes.

Moreover, one federal appellate court has already held very similar conduct to be infringing preparation of a

Third-party upgrade

In *Hubco Data Prods. Co.* v. *Management Assistance, Inc.*, 219 USPQ 450 (D. Idaho 1983), Hubco, the copyright owner, sold different versions of its computer program designed to serve computer systems having different amounts of memory. Hubco charged a higher price as the amount of memory handled increased. Hubco also sold upgrade services. MAI, the infringer, engaged in a competing upgrade service (not the sale of an add-on program as such), by modifying the code of installed Hubco programs to make them serve more memory. The court based its decision on legal theory that MAI infringed by reproducing copies of the copyrighted program in the course of decompilation and study undertaken to learn how to make appropriate upgrades.

derivative work. In *Midway Manufacturing Co.* v. *Artic International*,[2] the court found that use of speedup kits to change the operation of arcade video games violated section 106(2). The speedup kit made the game harder to play, to be sure, while the Game Genie makes the game easier, but that merely reflects different user purposes being served.

In the speedup case, arcade proprietors wanted to make the game harder because customers found it so easy that they either lost interest or lingered over it for an interminable time without inserting additional coins. The speedup kit therefore improved the revenue that arcade owners could earn from video game equipment they had purchased to earn revenue.

In the present case, some users find the game too hard to enjoy playing. They therefore want to improve their enjoyment from the game which they (or their parent) purchased to provide them with home entertainment. Whether the modification makes the game harder or easier does not meaningfully change the fact pattern of the Artic case from that of the Galoob case. The key fact is that the alleged infringer's conduct alters the code and display.

No fixed copy. The Galoob court made a point of the lack of a tangible copy of the modified game program. It noted that the modified version would not be transferrable to a third person because there was no copy. But that would be relevant only if some other section—not section 106(2)—were involved. Section 106(1) prohibits making unauthorized copies. Section 106(3) prohibits transferring unauthorized copies to others. But this case did not allege a copyright infringement under those sections of the Copyright Act.

The court sought to support its ruling by the statutory wording of the definition of derivative work, which includes the phrase "or any other form in which a work may be recast, transformed, or adapted." It said that "form" implied fixed and tangible form. The court was not made aware of the legislative history of the definition, which is contrary to the court's theory.

The statute is a mess. But it is not the trial court's job to rewrite it.

The House Report accompanying the 1976 Copyright Act points out that the omission in section 106(2) of any requirement of fixation in a tangible form was intentional. Section 106's antireproduction and antidistribution clauses contain fixation-in-copy requirements, but that requirement was deliberately omitted from section 106(2)'s provision against preparation of a derivative work—albeit for rather frivolous reasons.

The report explained that the forms of some copyrightable works lend themselves to preparation of derivative works in impermanent or intangible form. Yet they deserved protection against unauthorized takings, which should therefore be defined as infringements. Congress cited pantomime and ballet as examples illustrating the claimed need to eliminate the fixed-copy requirement for infringement by making derivative works. To save pantomime from piratical derivative works, therefore, Congress made preparation of derivative-work versions of pantomimes—and all other works—a copyright infringement, regardless of whether a tangible copy was made. Indeed, Congress did not even require as a condition of infringement liability that anything be done with the unauthorized derivative work.

Congress may have made an unwise or even foolish decision. It should have limited liability for preparing derivative works in intangible form to pantomimes and similar works, so the statutory remedy would not sweep up conduct unrelated to its legislative concern. At least Congress should have required some kind of use after preparation before liability attached. The statute is a mess. But that does not mean that the trial court should rewrite it to correct the legislative error. That is not its job under our legal system.

Users' rights. In further support of its construction of the phrase "derivative work," the Galoob court pointed to the nature of the competing interests at stake. It said the copyright law's purpose is to balance "a fair return on an author's creative labor against the need for 'broad public availability of literature, music, and the arts.' " Galoob sells Game Genie to users who have already paid Nintendo its price for the video game cartridge. Users modify the games only for personal enjoyment, not for commercial gain. The conduct is analogous to skipping commercials on a videotape of a television program by fast-forwarding, or rewinding and viewing in slow motion a critical play of a football game. None of this, the court said, deprives the copyright owner of the opportunity to derive "current or expected revenue."

(The facts are somewhat more complicated than the court paints them, although on balance its assertion may accurately characterize them. The court did not mention here that Nintendo sought to gain added revenue from its customers by marketing somewhat different game modification devices to them. One could argue, therefore, that to whatever extent Galoob satisfies this market, Nintendo cannot instead satisfy it for its own profit and is therefore deprived of expected revenue. In response, Galoob might make two points. First, to date Nintendo has neglected the needs of these particular customers and left them unsatisfied or else they would not be Game Genie customers. Second, why should Nintendo have a monopoly over this ancillary market? That is something to be decided, not assumed.)

The court summed up the equities of the case as follows: "Having paid Nintendo a fair return, the consumer may experiment with the product and create new variations of play, for personal enjoyment, without creating a derivative work .... For these reasons, this Court finds that the Game Genie does not create a derivative work protected by the copyright laws."

The result may be correct, but the legal analysis has gaps. The court's argument is sound for creation of a defense of estoppel, privilege, or implied or constructive license. But such an affirmative defense is quite distinct from the proper statutory construction of the phrase "derivative work."

Fair use. As an alternative holding, and assuming in the course of the argument that Game Genie prepared a derivative work, the court went on to find a fair use. Fair use is a statutory privilege codifying a series of judicial decisions favoring certain uses of a copyrighted work as privileged or immunized from liability. The privilege is the end user's in the first instance. But it extends to a person charged with contributory infringement liability because of responsibility for an end user's conduct—for example, if the person sold the end user the equipment used to commit the alleged copyright infringement. There can be no contributory infringement without direct infringement. Hence, if the end user's alleged direct infringement is privileged as fair use, then the accused contributory infringer cannot be punished in damages for contributing to a nonexistent direct infringement.

That the end user's use is noncommercial creates a rebuttable presumption in favor of fair use. Here, the end user uses Game Genie for private home entertainment. This places the facts of the case on a par with those of *Sony Corp.* v. *Universal City Studios, Inc.*[3] In that case, the Supreme Court found it fair use for users of home videotape recorders to record broadcasts for later viewing.

If the user's conduct is privileged as fair use, the supplier cannot be punished for contributing to a nonexistent infringement.

Another factor in favor of a finding of Game Genie fair use, the court concluded, was that the end users had already paid Nintendo for the video game cartridge. The court felt that purchase of the game cartridges gave customers a right to maximize their enjoyment of their purchase, including by modifying how the product worked.

Finally, the most important factor in the fair-use analysis was that Nintendo could not show that Game Genie would adversely affect the market for the copyrighted work by diverting sales away from Nintendo. The court rejected Nintendo's claim that Game Genie harmed the "Nintendo Culture," a concept the company promoted as "the apex of Nintendo's marketing strategy ... a [customer] mind-set intentionally created by Nintendo." Part of the Nintendo Culture, according to its marketing experts, is peer rivalry among video game players, who gain prestige by achieving high scores in the game. Players verify their achievement by photographing a screen displaying the high score. If everyone could get high scores with Game Genie, Nintendo said, then "this socially reinforcing practice would fall by the wayside," and Nintendo would lose future sales.

The court refused to believe this theory because Nintendo was itself marketing products and a magazine that helped players modify game play in a manner similar to Game Genie. In any event, the court would not award Nintendo in litigation what Nintendo could not achieve by its competitive efforts in the marketplace–"the exclusive right to modify game play as it alone sees fit"–because the Copyright Act does not bestow that power on copyright owners.

Implications for add-on programs. Much of what the court said carries over from patching video game computer programs with a Game Genie to patching an application with an add-on program. First, the modified code exists temporarily in RAM and is not written into ROM. That, of course, disregards the peculiar history of the part of the Copyright Act dealing with copyright infringement by preparation of derivative works.

More important, the modification occurs for the benefit of an end user who has already purchased a copy of the underlying copyrighted computer program. By the same token, the copyright owner has already been paid once for the right to use the program. The end use may be for commercial or noncommercial purposes, depending on the user. No one sells or transfers the modified program to others. Finally, one can assert that use of the add-on program will not divert sales away from the copyright owner.

These factors, on balance, suggest that the verdict in an add-on copyright infringement suit should be against the copyright owner and in favor of end-user rights. But the chain of argument summarized above has defects which might lead a different court to reach a different result. On the other hand, the

Galoob court left unmentioned other arguments favoring add-ons and end users' rights which, if properly presented, might lead another court to the same result by another route.

Alternative analyses. The Galoob court's opinion is a dog's breakfast. First, a derivative work was probably prepared, even though it was not a fixed, tangible copy. The existance of such a copy indicates the need to consider possible affirmative defenses (fair use is only one) that may negative the case of copyright infringement. Even when a copyright is infringed, sometimes circumstances excuse the infringement.

There may well have been a fair use, but the court's legal analysis of fair use is flawed by its overstatement of the case. Such overstatement is inherent in fair-use analysis. The legal standard for determining whether a use is fair calls for a balance of four incommensurable factors, in the fom, fair use = apples + oranges + lemons + grapes. The only way the court felt it could balance the factors against one another was to say that none of them favored the losing party. Otherwise, the court would have been compelled to decide whether the apples carried more legal weight than the oranges—a daunting task.

(By way of analogy, to find a minimum or maximum of $F(x,y,z,t)$, you must solve for the partial derivatives of F with respect to x, y, z, and t each successively being set to zero. Otherwise, you cannot tell that you have a peak rather than merely a point along a ridge or a saddle point.)

The court defined away the problem by assuming that Game Genie sales were not diverting sales of Nintendo's copyrighted product. That tipped the most important of the fair-use factors wholly in Galoob's favor. But Nintendo was, by the court's own account, merchandising a set of ways to prepare derivative-work versions of the copyrighted video games, while Game Genie provided another. Arguably, this resulted in competing versions of the copyrighted work in the same marketplace. That fact, if it is one, casts doubt on the correctness of the court's conclusion that the alleged copyright infringement did not supplant Nintendo to any degree as a seller of the copyrighted work.

An add-on program can dry up the market for later versions.

The same kind of thing can occur with an add-on program. Consider 4DOS, a shareware add-on program that interacts with MS-DOS. 4DOS, which has been available for several years, offers its users many functions that Microsoft did not include in MS-DOS 2 and 3—for example, on-line help with DOS command meanings and ability to recall and edit prior command entries (by using arrow keys). I am in no hurry to upgrade to MS-DOS 5, since 4DOS already provides most of what I would get from it (and most or all of the rest is found in old, already-installed programs such as 1Dir+ and PC Tools). An add-on program can thus dry up the market for later versions of the underlying program.

That does not necessarily tip the fair-use analysis in Nintendo's favor or against add-on programs in general. But the need to attempt an apples versus oranges fair-use analysis, instead of defining it away, makes the fair-use approach much more precarious. An alternative legal analysis may therefore be preferable, if it supports the same result.

The court's repeated emphasis on user rights points the way toward several possible such analyses. One is the legal doctrine of implied license. In patent law, a purchaser of a patented product has an implied license to modify it to increase its value. For example, the purchaser of a canning machine may modify it to process a different size can. The implication of the license is apparently by action of law, not from the surrounding facts. That is, the court implies the license based on its sense of fitness, not on the basis that the parties actually intended to agree to a license. That suggests that an attempted disclaimer of the implied license by the patent owner would be ineffective. The argument from patent law has not yet been carried over to copyright law. It may fail, but it is worth considering. Implied license would seem to be at least as strong an argument as fair use.

A related alternative legal argument is estoppel: The seller is estopped from preventing the customer from fully enjoying the use of purchase. There appears to be no substantial difference between estoppel and implied license. Estoppel is just another name for the idea that, by selling the product to the customer, the seller has acted in a manner inconsistent with preventing the customer from fully benefiting from the sale.

The doctrine against derogation from grants provides yet another legal argument amounting to the same thing. In *British Leyland Motor Corp.* v. *Armstrong Patents Co.*,[4] the House of Lords held that a car manufacturer, who owned copyrights covering tail pipes and other spare parts, could not require car owners to procure spare parts only from its licensees. The court considered that to use copyright law for this purpose would derogate from the title to the car that the manufacturer had conveyed to the customers upon sale. Any added expense or inconvenience imposed on car purchasers that interfered with their enjoyment of the purchased goods would derogate from the grant of title. Therefore, the court would not permit the seller to assert its copyrights to cause such results.

Finally, probably the strongest alternative legal argument, one wholly unmentioned by the Galoob court, could be based on section 117 of the Copyright Act. Section 117 gives owners of copies of computer programs a right to make adaptations of the programs when they do so as "an essential step" in their utilization of the computer programs. The NES console is a computer, within any reasonable definition of that term. It is a low-end, special-purpose microcomputer having a microprocessor chip as central processing unit. The video game cartridge contains a stored computer program, among other things. At least prima facie, the fact situation is that defined by section 117.

The only problem areas are

- Does the fact that modifying the computer program also modifies an audiovisual work take the case outside section 117?
- Is the adaptation "an essential step" in utilization of the computer program?

On the first point, a court would probably regard the program and audiovisual display as a unitary work. That is how the Copyright Office registers them. Therefore, the right to modify the program should carry with it the right to let the modifications change the audiovisual display since the two are a single legal unit.

The second point is less predictable. Legal authority is divided, but the better view is that "an essential step" is one intrinsic to the contemplated use of the program or one that the end user strongly desires to accomplish.[5]

These alternative rationales for the Galoob court's result, singly or in combination, would probably have given stronger support to the judgment than the fair-use analysis did. They are not different in kind, however, from the court's rationale of focus on users' rights. Where the proposed alternatives differ from the court's approach is that they seek to provide a legal theory centrally based on users' rights rather than to invoke such rights as incidental support for another legal theory.

One may quarrel with the chain of legal reasoning in the Galoob decision, but Galoob definitely tells us something important. It suggests that courts favor the rights of the customer in an add-on situation, at least when copyright owners cannot show that the customer is taking a free ride at their expense. As Justice Black once said about repair and reconstruction of patented products, "One royalty to one patentee for one sale is enough."[6]

Courts are likely, therefore, to feel that customers deserve considerable freedom to modify a purchased computer program. The sentiments about relative equities expressed in the Galoob decision may thus be more precedential, for prediction purposes, than the legal analysis.

References

1. 20 USPQ 2d 1662 (N.D. Cal. 1991) (12 July 91).
2. 704 F. 2d 1009 (7th Cir.), cert. denied, 464 US 823 (1983).
3. 464 US 417 (1984).
4. [1986] 1 All Eng. Rep. 852 (H.L.).
5. Foresight Resources Corp. *v.* Pfortmiller, 13 USPQ 2d 1721 (D. Kan. 1989).
6. Aro Mfg. Co. *v.* Convertible Top Replacement Co., 365 US 336, 360 (1961) (concurring opinion).

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 177 Medium 178 High 179

Micro Review

continued from p. 7

I installed Disk Express II and watched it go through its paces. I don't think my hard disk was badly fragmented, so I'm not sure how much improvement I've seen, but I am sure of two things. Disk Express II is easy to use, and I feel a lot better knowing that its many features will be available if odd situations arise.

Multidisk is a hard-disk partitioning program. It allows you to place files into logical groups kept together on your disk to minimize access times. A number of protection features provide advantages on networked systems or on systems that more than one person can access.

Alsoft also claims that partitioning your disk provides another layer of protection against computer viruses. I have not had virus problems on my Mac, but my PC was recently infested by the notorious Michelangelo virus. That's a story for another column.

Disk Check diagnoses disk and directory problems. It couldn't find any problems with my disk or directories, but it did help me identify and remove an invisible anchored file belonging to a program I got rid of long ago.

If you use your Macintosh regularly, Disk Express II will certainly improve your efficiency. I think it's worth the modest price.

I don't have space to describe all of the other interesting programs I've received recently. Some of these will appear in future columns.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

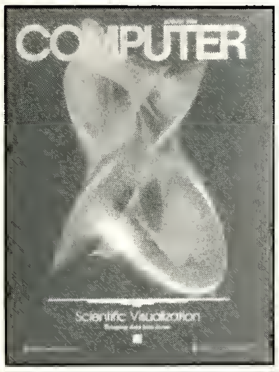
Low 183 Medium 184 High 185

IEEE COMPUTER SOCIETY

1992 Membership/Subscription Application

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

BENEFITS



Receive *Computer* automatically with membership. Written and refereed by experts, it features articles on the latest developments in computer technology, applications, and research, as well as survey and tutorial articles on topics covering the entire computer field.

Computer's popular departments include New Products, Product Reviews, Book Reviews, Standards, Open Channel (a reader forum), and Computer Society News.

Other membership benefits include the following:

- Discounts of up to 50% on **Computer Society Press books and videos.** Over 700 titles covering a broad spectrum of computer science topics including networking, communications, advanced systems, image processing, security, artificial intelligence, and visualization
- Discounts on registration to over 100 conferences, workshops, and tutorials each year
- Low subscription rates on special-interest periodicals
- Opportunity to participate in any of our 32 technical committees—networks of professionals with common interests in computer hardware, software, and applications
- Opportunity to participate in the development of more than 200 standards projects sponsored by the society
- Low member rates on COMPMAIL, the electronic mail service especially designed for computer professionals

Schedule of Fees

To join: see item 1, 2, or 3.
To subscribe: see item 4.

Membership dues and periodical subscriptions are annualized to, and expire on, December 31. Pay full- or half-year rate depending on date of receipt by the Computer Society as indicated below.

PRICES EXPIRE 12/31/92

		Half Year Mar 1-Aug 31	Full Year Sept 1-Feb 28
1	I don't belong to the IEEE and I want to join just the Computer Society	☐ $27	☐ $ 54
2	I want to join both the Computer Society and the IEEE*		
	I reside in Region 1-6 (United States)	☐ $58.50	☐ $117
	I reside in Region 7 (Canada)	☐ $53.50	☐ $107
	I reside in Region 8 (Europe, Africa, or the Middle East)	☐ $53.00	☐ $106
	I reside in Region 9 (Latin America)	☐ $49.50	☐ $ 99
	I reside in Region 10 (Asia and Pacific)	☐ $48.50	☐ $ 97

*ACM members who join both IEEE and the Computer Society may deduct $5 off the full-year rate; $2.50 off the half-year rate.

		Half Year	Full Year
3	I already belong to the IEEE and I want to join the Computer Society	☐ $11.00	☐ $22

IEEE Member Number ____________________

4 SPECIAL-INTEREST PERIODICALS for new or current members

	issues per year	Half Year	Full Year
IEEE Computer Graphics and Applications	6	☐ $12.50	☐ $ 25
IEEE Design and Test	4	☐ $11.00	☐ $ 22
IEEE Expert	6	☐ $10.00	☐ $ 20
IEEE Micro	6	☐ $11.50	☐ $ 23
IEEE Software	6	☐ $12.50	☐ $ 25
Transactions on:			
Computers	12	☐ $12.00	☐ $ 24
Knowledge and Data Engineering	6	☐ $ 9.50	☐ $ 19
Parallel and Distributed Systems	6	☐ $ 9.50	☐ $ 19
Pattern Analysis and Machine Intelligence	12	☐ $12.00	☐ $ 24
Software Engineering	12	☐ $11.00	☐ $ 22
IEEE Annals of the History of Computing	4	☐ $ 8.00	☐ $ 16

☐ Visa ☐ MasterCard ☐ American Express

Charge Card Number (Minimum Charge $10)

Mo. Yr.
Exp. Date

Checks drawn in local currency on a bank in the country of origin accepted from members in Australia, Austria, Belgium, Canada, Denmark, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, UK, USA, and Yugoslavia.

Periodicals total $________

CA, DC residents add applicable sales tax $________

Membership fees $________

Total $________

CANADIAN residents add 7% GST; BELGIAN residents add 6% VAT $________

Total enclosed $________

I hereby make application for Computer Society membership and, if elected, will be governed by IEEE's and the society's constitutions, bylaws, and statements of policies and procedures.

MAILING ADDRESS

Full signature ________ Date ________

☐ Male ☐ Female

First name ________ Middle initial(s) ________ Last name ________ Date of birth ________

Street address ________ City ________ State/Country ________ Zip ________

EDUCATION (highest level completed) Course ________ Degrees received ________ Date ________

Name of educational institution ________

OCCUPATION ________
Title or Position

Firm name ________

Firm address ________

City ________ State/Country ________ Zip ________

ENDORSER (an IEEE member, Senior Member, or Fellow)

Endorser's signature ________

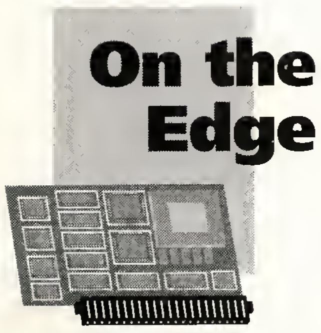
Name (print in full) ________ IEEE Member No. ________

Street address ________

City ________ State/Country ________ Zip ________

Mail or fax to appropriate Computer Society office:

EUROPE: 13, Avenue de l'Aquilon, B-1200, Brussels, BELGIUM. Fax: 32-2-770-8505
PACIFIC RIM: Ooshima Building, 2-19-1 Minami-Aoyama, Minato-ku, Tokyo 107, JAPAN. Fax: 81-3-3408-3553
ALL OTHERS: 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264 USA. Fax: 714-821-4010



Carl Warren

McDonnell Douglas

Space Systems Company

(714) 896-3311

x. 7-1230

warren@ssdunx.mdc.com

Regression testability

[This issue, Lee White and Hareton Leung make an argument for a more systematic approach to regression testing in software development.

I invite readers to send information on a tool or method that solves problems, for consideration in future columns. – C.W.]

Lee White
Case Western Reserve University

Hareton K.N. Leung
Bell-Northern Research

Designers consider many criteria when writing software. In these times when change is so common, one of the most important is *maintainability*, which strongly correlates with other desirable criteria. We propose a more systematic approach to an important aspect of maintainability: regression testing.

We can differentiate between *perfective* maintenance, which enhances software functionality, and *corrective* maintenance, which detects and corrects defects. Whether the maintenance is perfective or corrective, we must ensure that it does not inadvertently affect unmodified functionalities. When this occurs, we call it a *regression error*. Regression testing uses test data previously developed for these unmodified functionalities to detect such regression errors.

Regression testing is important at the unit, integration, and system (or functional) testing levels. However, software development teams usually have responsibility for unit and integration testing. These teams do not consistently apply regression testing at these levels when they make changes and often do not even systematically retain test data. System or functional testers, on the other hand, are very systematic about keeping test data and applying regression testing. This costs more than detecting regression error earlier.

We recently completed a research project,[1,2] where for small changes we endeavored to identify subsets of test data to use as regression tests at the unit, integration, and system levels. Our idea was to cut the cost and time to run the regression tests for numerous small changes in the software and to focus on the areas of tests related to code or functionalities of the change.

Figure 1 illustrates the approach for regression testing at the unit level so a subset of the test data can detect a unit regression error. The static analyzer detects those test data, called reusable tests, that cannot be affected by the program changes. We could accomplish this detection with static slices, as proposed by Weiser,[3] which identify statements that could be affected by program changes. If this fails to lead to a sufficient reduction in test data, we could use dynamic slicing, as introduced in Korel and Laski,[4] which indicates statements actually affected by program changes.

The dynamic analyzer executes the remaining tests and identifies obsolete tests that no longer achieve their intended function since the program has changed. The remaining testable tests reveal regression errors. We require new tests to update functional tests if the specification of the module has changed, or to obtain structural tests to achieve a specified level of coverage. To do so, we can use the module dynamic tester shown in Figure 1.

The method we propose may result in a higher payoff situation for regression testing at the integration level. We endeavored to develop a general approach that does not depend on the particular type of integration (such as top-down, bottom-up, or hybrid), as long as it is incremental.

Firewall. A key question is, given the modified modules, how many modules must we retest? In other words, where can we draw a *firewall* around those modules which must be retested? When we detect errors, we should make changes so as to keep the firewall from spreading to include more system modules if possible.

To make this analysis precise, we assume that all module dependencies are indicated in the call graph. Figure 2 shows an example call graph with four modified modules C_1-C_4. This is a severe assumption because global variables are another common dependency in many programs, in which a number of modules may define global variables used by otherwise unrelated modules. I will briefly discuss global variable testing later.

Resource dependencies may exist between modules. An example is memory, in which the resource is constrained between a number of modules. We assume to indicate a precise result, the only errors present are due to the modified modules from the maintenance effort. We also assume reliability of the unit and integration tests. (I will return to this assumption at the end of this analysis.)

Given these assumptions, we analyzed a number of basis cases describing module dependencies in the call graph.[2] The possibilities for a module *a* are

- no change in *a*, NoCh(*a*),
- only code change in *a*, CodeCh(*a*), and
- change in the specification of *a*, SpecCh(*a*).

If module *a* calls module *b*, then we can ignore any cases in which neither *a* or *b* is changed. This leaves us with eight cases to consider. We must also model the addition of new modules and deletion of modules for which we have identified several more cases.

Figure 3 on the next page indicates the four critical boundary cases. Two cases correspond to an unchanged module calling a modified module, and

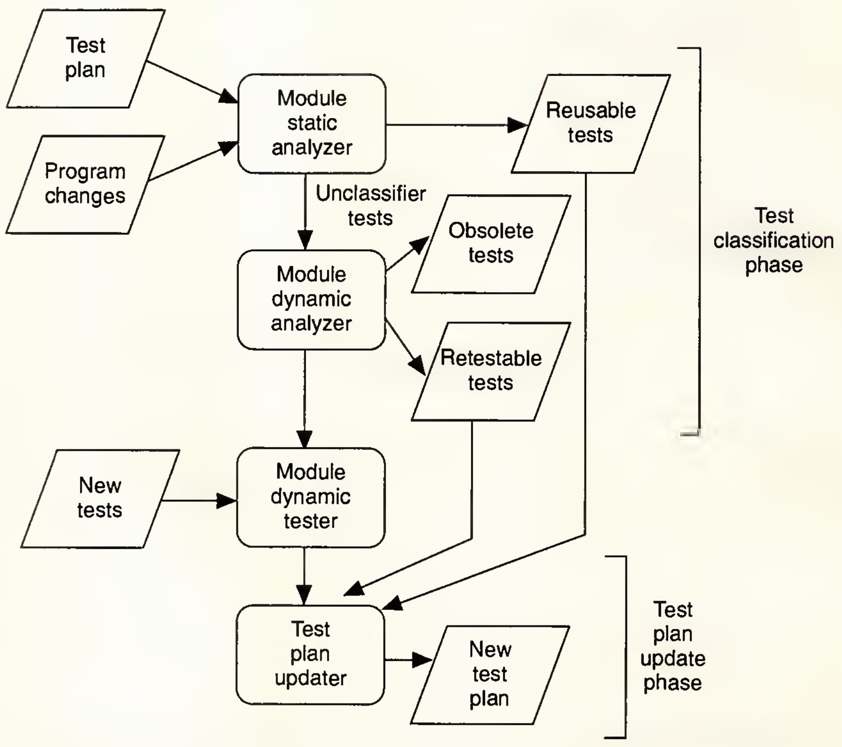


Figure 1. Unit regression testing.

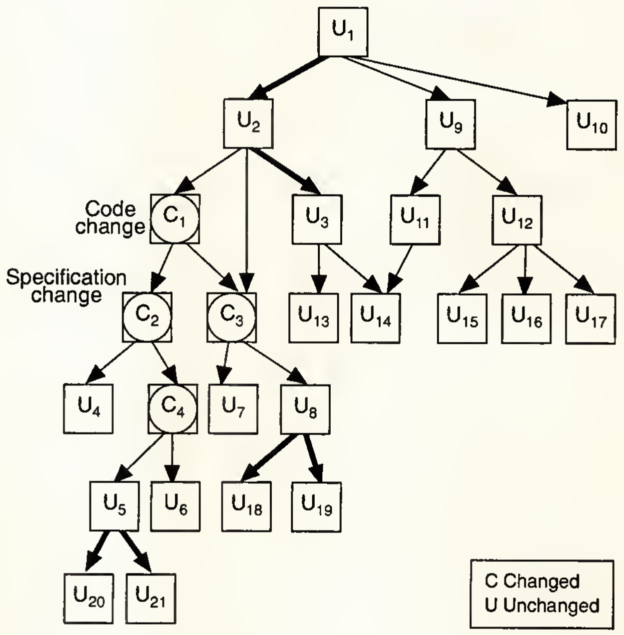


Figure 2. Basis cases and the calculation of a firewall, indicated by bold arrows.

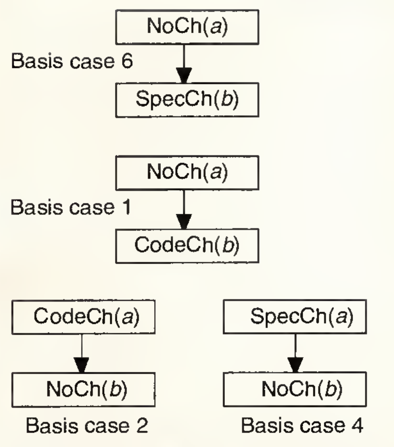


Figure 3. Boundary cases for firewall construction.

the other two correspond to a modified module calling an unmodified module. Case 6 in Figure 3 is unusual in that one would not expect an unchanged module to call a module in which the specification has changed. This unusual situation creates two consequences.

- We should reexecute not only the integration tests between modules but also the unit test of the calling module.
- Case 6 is the only case in which we cannot guarantee that only the modified module needs to be changed if any tests detect an error. If the calling module is modified, the firewall expands.

The other three cases are simpler and can be characterized by the following observations:

- We need to reexecute only the integration tests between the two modules in each case. We do not have to rerun the unit tests for the unchanged module.
- If any of the tests detects an error, we can correct the error by changing only the modified module. The programmer should not change the unmodified module. If we follow this discipline, the firewall does not expand.

To illustrate the firewall concept, return to Figure 2. The four modules C_1-C_4 are given as modified. We must rerun integration tests U_2-C_1, U_2-C_3, C_2-U_4, C_3-U_7, C_3-U_8, C_4-U_5, and C_4-U_6, but no unit tests for unchanged modules need be rerun. Figure 2 shows the firewall as bold arrows that separate the affected modules from the rest of the call graph—just as the firewall does in real testing.

I must make a final remark about our assumption that no errors exist in the system other than those due to the modified modules and that all unit and integration tests are reliable. Of course these assumptions do not hold true in practice, and thus our precise conclusions are no longer valid. However, we can make these conclusions practical by observing that testing the modules within the firewall is a sensible use of testing resources, even if we cannot rule out errors existing elsewhere.

Computational experience. We also conducted an experiment to evaluate these regression testing concepts.[2] We used a student database program with 20 distinct modules, 32 modules, seven software features (major software functions in the specifications), and over 550 executable lines of Pascal code. The author provided four real modifications that we could evaluate with regression testing using reduced tests.

Table 1 shows the results of this study. Modifications 1, 2, and 4 show considerable reductions in the number of required tests, but modification 3 shows little reduction. The top four lines in Table 1 show the reason for this. Modification 3 has a slightly higher number of affected modules or module interactions than the other modifications. The biggest difference is in the number of affected features. Since modification 3 affects all features, the number of system tests does not decrease. Note that the design was good

Table 1. Regression testing evaluation.

	Modification			
	1	2	3	4
Affected source lines	25	80	23	57
Affected modules	2	4	8	8
Affected module interactions	2	3	16	10
Affected features	1	1	7	1
Regression tests				
Unit	15	22	50	40
Integration	32	32	120	80
System	46	24	130	38
Total tests				
Unit	27	40	67	66
Integration	246	278	275	307
System	106	130	130	158
Total tests	379	448	472	531
Regression tests	93	78	300	158
Percentage	24%	17%	63%	30%

enough to anticipate modifications 1, 2, and 4 but was not maintainable for modification 3.

To establish the effectiveness of the test subsets and find which tests detected errors, we asked the programmer to introduce 13 logic errors. Of these, the tests detected 12, with the subsets as effective at detecting these errors as the full test sets. Of the 12 errors detected,

- unit testing detected eight errors,
- integration testing detected 11 errors, and
- system testing detected 12 errors.

Some lessons here mirror current practice. We could avoid bothering with unit or integration testing and conduct only system testing, but this would be very expensive and time-consuming. The values of both unit testing and integration testing are clear. Integration testing detects different errors than does unit testing. Developers should not only perform both types of testing but also save the data to do regression testing at these two levels.

Global variables. We also studied regression testing global variables[5] and found that the global variables may be treated as parameters passed between modules for the purpose of regression testing. This is despite the fact that global variables are insidious in that many modules may become dependent through extensive use of global variables. A change in one or a few modules may affect change in modules throughout the entire system.

We are completing research on how developers should test global variables. The study is complex, but should provide an approach for developers to actually test the effects of global variables they insist on using.

References

1. H.K.N. Leung and L. White, "Insights into Regression Testing," *Proc. Conf. Software Maintenance*, IEEE CS Press, Los Alamitos, Calif., Oct. 1989, pp. 60-69.
2. H.K.N. Leung and L. White, "A Study of Integration Testing and Software Regression at the Integration Level," *Proc. Conf. Software Maintenance*, IEEE CS Press, Nov. 1990, pp. 290-301.
3. M. Weiser, "Programmers Use Slices When Debugging," *Comm. of ACM*, Vol. 25, July 1982, pp. 446-452.
4. B. Korel and J. Laski, "Dynamic Program Slicing," *Information Processing Letters*, Oct. 1988, pp. 155-163.
5. H.K.N. Leung and L. White, "Insights into Testing and Regression Testing Global Variables," *Software Mainten-ance: Research and Practice*, John Wiley & Sons, Sussex, UK, Vol. 2, 1990, pp 209-222.

Lee White chairs the Department of Computer Engineering and Science at Case Western Reserve University in Cleveland, Ohio. His research interests include software testing, verification, and algorithm analysis.

White earned a BSEE from the University of Cincinnati and MS and PhD degrees in electrical engineering from the University of Michigan.

Hareton K.N. Leung works for Bell-Northern Research while he is completing his PhD work in computing science at the University of Alberta. His interests include test strategy development, process improvement, quality assurance, and software reliability engineering.

Leung earned a BS in physics and astronomy from the University of British Columbia and an MS in computing science from Simon Fraser University.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 192 Medium 193 High 194

1992 Gordon Bell Prize

For Outstanding Achievements in the Application of Parallel Processing to Scientific and Engineering Problems

Entries are due **May 1, 1992,** with finalists to be announced by June 30 and winners announced at the Supercomputing 92 conference in November 1992. Prizes of $1,000 each will be awarded in two of three categories:

- Performance, based on megaflop rate on a machine with known performance compared against similar applications. If this is not possible, entrants should document their performance claims.
- Price/performance, based on performance divided by the cost of the smallest practical computational engine, including critical peripherals. Performance measurements will be evaluated as for the performance category.
- Compiler parallelization, based on the most speedup, measured by dividing the wall-clock time of the parallel run by that of a good serial implementation of the same job.

General conditions include demonstrating the utility of the program and machine. The judges will also consider how much the entry advances the state of the art in some field.

For more information or to enter, contact:

1992 Gordon Bell Prize
c/o Marilyn Potes
IEEE Computer Society
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264
Phone: (714) 821-8380

IEEE MICRO

Editorial Calendar

JUNE 1992

Associative memories and processors

- Late-breaking developments in wide-word content-addressed memories (CAMs)
- Dynamic CAPP (parallel processor) architectures
- Fault-tolerant architectures for crucial control systems such as those in trains, space systems, etc.

Ad closing date: May 1

DECEMBER 1992

Special signal processors

- Recent information from the vital area of digital signal processors
- News about other techniques for signal processing
- Mixed analog/digital processors solve a real need
- Neural networks process signals following methods used by the human brain

Ad closing date: November 1

AUGUST 1992

European industry

- Recent developments in integrated circuit and microsystem technology from major European manufacturers

Ad closing date: July 1

FEBRUARY 1993

Antomotive/traffic microelectronics

- Worldwide developments in microelectronics for traffic and driving assistance
- Latest developments in the European Prometheus and US IVHS programs
- News from Japan too
- Improving traffic safety with electronics

Ad closing date: January 2

OCTOBER 1992

Processing hardware for video communication

- ICs for HDTV compression
- ICs for HDTV communication
- Parallel processing systems with real-time video compression capabilities
- Multimedia systems with real-time video compression capabilities

Ad closing date: September 1

APRIL 1993

Advanced packaging/interconnect technology

- Critical trends and issues
- Flexible, glass, or diamond substrates
- Few-chip or 3D packaging
- Attachment, bonding, and connection technologies including fine-pitch surface mount, laser applications, known-good die, and interconnection trade-off analysis

Ad closing date: March 1

IEEE Micro helps designers and users of microprocessor and microcomputer systems explore the latest technologies to achieve business and research objectives. Feature articles in *IEEE Micro* reflect original works relating to the design, performance, or application of microprocessors and microcomputers.
All manuscripts are subject to a peer-review process consistent with professional-level technical publications. *IEEE Micro* is a bimonthly publication of the IEEE Computer Society.

Advertising information: Contact Marian Tibayan, Advertising Department, IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380; fax (714) 821-4010.

Articles may change. Please contact editor to confirm.

Micro News

Send information for inclusion in Micro News one month before cover date to Managing Editor, IEEE Micro, PO Box 3014, Los Alamitos, CA 90720-1264.

Second-generation RISC chips

Ware Myers, Contributing Editor

Authors presented papers describing 10 second-generation RISC processors at Compcon Spring 92, in February. Table 1 lists the papers, which are contained in the Digest of Papers, available from the Computer Society.

Most of these papers address new or unusual design features, so they do not contain complete application data. The most advanced chips have line widths of 0.75 to 0.80 μm and contain more than one million transistors. Most of them achieve high performance by using both superscalar and superpipeline techniques. With a million plus transistors the chip can, of course, contain many performance-enhancing features.

DEC Alpha. The Alpha is not merely a new chip. It is a new architecture that DEC intends "to withstand the test of time" for the next 25 years. EV4 is the first implementation of this architecture. DEC envisions more powerful implementations in coming years. "Future generations will be able to deliver up to a 1,000-fold increase in performance," the company said in its announcement.

The first implementation of Alpha is in 0.75-μm, 3.3V, CMOS technology. It contains 1.68 million transistors on a 1.68 × 1.39-cm chip with 431 pins and operates at 200 MHz. The 64-bit CPU can issue two instructions each clock cycle to two of the four pipelined functional units: integer, floating point, branch, and load/store. Thus, the peak issue rate can reach 400 MIPS.

Alpha's architecture accommodates Digital's Open Advantage by supporting both OSF/1 and Open VMS operating systems. Thus, it provides an upgrade path from DEC's existing VAX architecture and an opportunity for any organization using Unix. DEC plans to license it to anyone.

Alpha can be employed as a single CPU in personal workstations, or in large aggregations it can form massively parallel systems. Cray Research plans to use it in this way.

HP reduces path length. "The path length of a computation is the number of instructions needed to process the computation," Ruby Lee of Hewlett-Packard's Cupertino, California, facility, pointed out in the session devoted to HP's PA (precision architecture) RISC architecture. "The execution time of a computation is the product of the path length in instructions executed, the average cycles taken per instruction, and the cycle time of the processor."

It follows then that we can improve the performance of a processor if we can reduce one or more of these variables. Cycle time depends largely on the underlying technology. In the case of HP's current implementation, clock time is down to less than 10 ns. Superscalar and superpipelining reduce the cycles per instruction. RISC architecture originally speeded up processing by limiting the instruction set to relatively fast-executing instructions, permitting clock time to be reduced.

According to Lee, "The PA-RISC approach is the first to specify a datapath to meet minimum requirements, then to find opportunities to exercise multiple independent functional units with a single instruction so as to minimize path lengths and improve cost performance."

This opportunity is found in "multi-op" instructions that combine two or three operations in the 32-bit instruction after the fashion of VLIW (very long instruction word) architecture. A single instruction, thus, can initiate several operations on separate hardware resources in parallel. In effect, this technique manages to do more work in a single instruction, shortening the path length of instructions implemented in this way.

First, the design team measured the frequency

0272-1732/92/0400-0086$03.00 © 1992 IEEE

Table 1. Papers on recent RISC microprocessors, as contained in the *Digest of Papers, Compcon Spring 92*.

Processor	Company	Lead author
PA-RISC 1.1	Hewlett-Packard	Eric DeLano
Supersparc	Sun Microsystems Texas Instruments	Greg Blanck
Pinnacle-1 (Sparc-compatible)	Cypress/Ross Technologies	Raju Vegesna
88110*	Motorola	Keith Diefendorff
Alpha (EV4)**	Digital Equipment	Richard Sites
NVAX**	Digital Equipment	Mike Uhler
PowerPC**	Apple/IBM/Motorola	Ron Hochsprung
i960 Cx	Intel	Elliot Garbus
Am29030/35	Advanced Micro Devices	Scott McMahon
LR33020 X-terminal controller	LSI Logic	Robert Tobias

*See page 40 in this issue.

** Extension of IBM RS6000 architecture

of occurrence of pairs or triples of operations. Then it selected several dozen to implement. "None of these multi-op instructions are difficult to generate from a compiler's point of view, since they map naturally to generic programming constructs," Lee noted. "The additional hardware required for these multi-op instructions is minimal compared to the additional performance provided."

Typically one of these multi-op PA-RISC instructions replaces two or three single-op instructions. In a few instances the replacement rate was in the range of five to 10 single-op instructions. "Overall, systems based on the PA-RISC architecture have achieved extremely competitive performance on both technical and commercial benchmarks," Lee concluded.

Metrology report

Measurement technology is the key to boosting semiconductor productivity, according to a US Department of Commerce report. *Metrology for the Semiconductor Industry* suggests that advances in metrology lead to breakthroughs in semiconductor technology. Manufacturers who are better able to detect defective chips—and to prevent defects from occurring—develop more efficient processes. Furthermore, as designers build smaller chips and incorporate an increasing number of transistors, the margin for error in manufacturing shrinks.

The federal report cites several sources contributing metrology technology that can help manufacturers keep pace with microprocessor research, including federal agencies, universities, corporations, and cooperative research groups. A free copy of the report is available from Jane Walters, B3444 Technology Building, National Institute of Standards and Technology, Gaithersburg, MD 20899.

Germany recycles old machines

A new law will require German manufacturers to take back old electronic equipment, beginning in 1994. The Electronic Waste Order aims at reducing the 800,000 metric tons of electronic waste that reach the country's incinerators and dump sites each year. Some manufacturers already take back worn-out equipment and dismantle it

Micro bits

Computer manufacturer **Silicon Graphics** will acquire **Mips Computer Systems**. Some analysts see the move as an attempt to keep the Mips architecture competitive in the race to control the brains of the next generation of personal computers. Meanwhile, **Intel,** whose 386 and 486 microprocessors form the basis for the current generation of personal computers, announced it had signed a letter of intent to share technology with **VLSI Technology**.

Cray Research and **Sun Microsystems** will share hardware and software technology to create a seamless environment for Sun's systems and Cray's supercomputers. Cray recently formed a subsidiary, Cray Research Superservers, to make and sell Sparc products and joined Sparc International, the consortium to promote scalable processor architecture. The supercomputer innovator plans to introduce a massively parallel system next year with a peak performance of 100 Gflops.

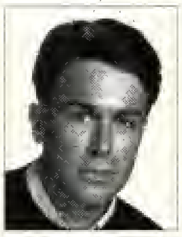
The neural networks in **Janus** translate spoken sentences into or from English, German, or Japanese. Carnegie Mellon, Siemens AG, ATR of Kyoto, and the University of Karlsruhe collaborated to build the 400-word, continuous speech system.

The University of New Mexico distributes **Khoros**, a software development environment for information processing and data visualization, at no charge through file transfer protocol sites. The distributed computing system, runs on Sun, DEC, IBM, Hewlett-Packard, Next, Mips, and Cray machines and is accessible on e-mail at pprg.eece.unm.edu.

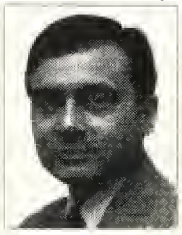
Three join editorial board

Editor-in-Chief Dante Del Corso has appointed three new members to the editorial board of *IEEE Micro*.

Teresa H. Meng is an assistant professor of electrical engineering at Stanford University. She will review manuscripts for the magazine. Meng earned a BS degree in electrical engineering at National Taiwan University and MS and PhD degrees in electrical engineering and computer science at the University of California, Berkeley.



Gilles Privat, a research engineer with France Telecom, the National Center for the Study of Telecommunications in Grenoble, will also review manuscripts. He heads a research group investigating areas of parallel algorithms and VLSI architecture for image processing. Privat earned engineering and doctoral degrees in signal and systems theory at Telecom Paris University.



Arun K. Sood is a professor of computer science at George Mason University. He will oversee plans for a new department that will feature short technical notes and "dream chips" submitted by readers. Sood received a bachelor's degree from the Indian Institute of Technology in Delhi and MS and PhD degrees from Carnegie Mellon University, all in electrical engineering. He is a member of the IEEE Systems, Man, and Cybernetics Society's Administrative Committee and recently guest edited *IEEE Micro's* theme issue on database machines.

to reuse parts. The new order requires them to recycle as much metal and plastic as possible. Precious metals will be salvaged and reused; other metals may be resmelted and used as slag in, for example, road paving.

Multimedia authoring program

A Stanford University programmer has developed software that lets students and professors create their own multimedia presentations with archive and original materials. From a Unix workstation, students can combine video and music with their own recorded commentary and typed-in text.

George Drapeau of the Academic Software Development Group of Stanford's Libraries and Information Resources created the program called Maestro (multimedia authoring environment). His prototype workstation includes a Sparcstation 2, microphone, laserdisc player, CD-ROM player for music and data, and stereo speakers.

The mouse-driven, icon-based interface allows users to access literary works available to on-line users of the university's networks. A video editor lets users choose segments, add music, record voice commentary, and type in text and captions. A time line editor designates how segments overlap.

Drapeau says the program is not designed to produce professional presentations, but to create a simple tool that lets users concentrate on the task and not the computer. He offers the program as "freeware" to students.

Current literature

The Glossary of Computer Security Technology defines security terms used by US federal departments and agencies. The glossary provides multiple definitions, reflecting various uses by different federal agencies. Technical information on the 176-page publication is available from Edward Roback at (301) 975-3696.

National Technical Information Service, Springfield, VA 22161; $26 (hard copy), $12.50 (microfiche).

The five-volume ninth edition of the *Index and Directory of Industry Standards* lists 113,000 standards from 380 organizations. Twelve-thousand standards are revised since the last edition; 8,000 are new. The directory cites standards by subject, society/numeric, society, and ANSI concordance. Volumes 1 and 2 comprise the US set, volumes 3, 4, and 5 are the international set.

Global Engineering Documents, PO Box 19539, Irvine, CA 92713-9539; $376 (complete set), $195 (US volumes only), $275 (international volumes).

The *Fall 1991 IEEE Standards Catalog Update* lists electrotechnology standards published since the institute issued its most recent catalog last September.

IEEE Standards, PO Box 1331, Piscataway, NJ 08855-1331; free.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 198 Medium 199 High 200

New Products

Send announcements of new microcomputer and microprocessor products to Managing Editor, IEEE Micro, *PO Box 3014, Los Alamitos, CA 90720-1264*

Joe Hootman

University of North Dakota

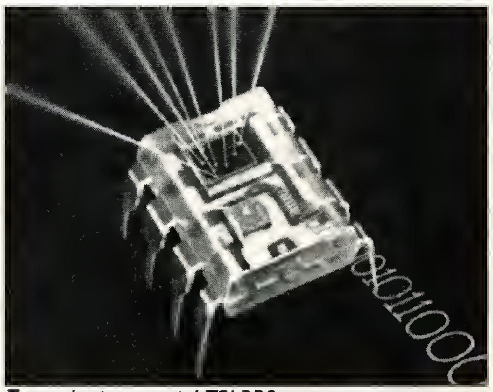
Devices and components

Light to frequency

For precision light measurements, the TSL220 converts light to digital signals. It comprises a photodiode and BiMOS current-to-frequency converter and connects directly to a microprocessor or a digital control circuit. The CMOS-compatible output voltage is a pulse train with a frequency directly proportional to light intensity of the diode.

The device features a dynamic range of 118 db and output levels of over 100 KHz in office desk lighting and as low as 1 Hz in the dark. The converter functions with a 5 to 10V power supply and in temperature ranges of –25° to 70°C. It is housed in an eight-pin, clear DIP package. *Texas Instruments; $4.61 (1,000s).*

Reader Service No. 10



Texas Instruments' TSL220

1-Mbit SRAM

PSM44039 is a processor-specific memory (PSM) chip that works as a secondary cache for high-end, synchronous RISC processors. The 1,179,648-bit chip is a self-timed, synchronous, CMOS SRAM organized as 128 Kwords by 9 bits. Available cycle times range from 15 to 25 ns. The PSM chip operates from a 5V supply in temperatures of 0° to 70°C. *Paradigm; $85 (20-ns version, 1,000s).*

Reader Service No. 11

10Base-T interface

The SN75LBC086 differential driver/receiver is a one-channel interface for concentrators, repeaters, and bridges in twisted-pair Ethernet systems. The 24-pin, 300-mil device features a squelch circuit for noise immunity beyond 10Base-T standard requirements. As also required by the standard, it provides jabber control, collision detection, signal quality, and link test functions. A loopback mode permits testing the data path while still connected to the network. *Texas Instruments; $8.40 (1,000s).*

Reader Service No. 12

One-chip display driver

For vacuum-fluorescent displays, the M66004 controller/driver generates 16 characters from RAM and 160 characters from ROM, for a variable display length up to 16 display digits in 5 × 7 segments. Two built-in static points drive LEDs or control peripheral ICs.

A three-line serial bus to the microcontroller receives data without needing a buffer. The chip also features an eight-step dimmer control, cursor display, and two scan-cycle formats. *Mitsubishi; $5.12 (1,000s).*

Reader Service No. 13

Pulse-width modulators

Designers can build 500-KHz, off-line power supplies and DC-to-DC converter applications with a line of pulse-width modulators. The LT1241-1245 modulators feature temperature-compensated reference, high-gain error amplifier, current-sensing comparator, 50-ns current sense delay, start-up current of less than 250 µA,

Neuron chip

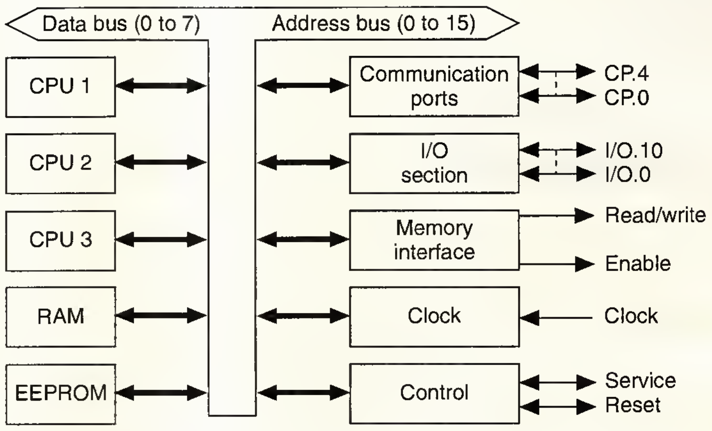
David Sims, Assistant Editor

Motorola's MC143150 Neuron Chip is a microcontroller with an embedded protocol that forms the heart of remote nodes in networks based on Echelon's Lontalk. Lontalk is a distributed sense and control network protocol for industrial, commercial, and residential systems that is specially designed to transfer small amounts of data, and thus reduce costs.

Because sense and control networks transfer relatively small packets of data compared to other network systems (for example, "Turn down the heat in the board room!" instead of "Here is the report on last quarter's production..."), they can get by with slower data transfer rates. Lontalk sends data packets of 15 bytes at up to 1.25 Mbps, considerably slower than Ethernet's 10 Mbps or some of the newer systems transmitting up to 150 Mbps.

According to Al Mouton, strategic planning manager in Motorola's Lonworks Products organization, the slower transfer rate is one way to reduce the costs of these systems. Another is the on-chip incorporation of all the functions needed to process inputs from sensors and control devices and respond with commands. Each Neuron Chip acts as an "intelligent controller" capable of continuing its monitor and command functions even if the network gets disconnected.

Mouton compared the difference between Lontalk and a centrally based system to the difference between desktop PCs and a mainframe system with "dumb" terminals.



Motorola's MC143150 Neuron Chip

"If a central computer system goes down, everyone just sits there staring at blank screens," he said. "With PCs on every desk, if the network fails, you can go on working. You just can't transfer data."

On-chip features include

- three 8-bit pipelined processors,
- an 11-pin I/O port programmable in 24 nodes,
- two 16-bit timer/counters,
- a five-pin communications port to support network transceivers,
- a 2-Kbyte SRAM,
- 512 bytes of EEPROM with charge pump,
- an external memory interface,
- a sleep mode, and
- a 48-bit ID number unique to each device.

Mouton also said that the on-chip incorporation of a protocol reduces the cost of each node. Echelon and Motorola hope that success for Lontalk will make it a de facto standard for local network command systems. Nodes within other systems that include the software, systems, and components to form a complete node can cost up to $50 per unit. Motorola expects, with volume production, to reduce the cost of the Neuron Chip to under $5 by 1994.

Given the unlimited number of nodes acceptable within Lontalk's hierarchical architecture, Mouton sees a wide range of potential applications, from manufacturing lines to home automation, from building security to systems in a recreational vehicle.

The Neuron Chip is packaged in a 64-pin quad flat pack. *Motorola; $11.78 (1,000s).*

Reader Service No. 14

and a high-current totem pole output stage suited to drive power MOSFETs. The chips incorporate blanking in the current sense comparator to prevent the leading edge current spike from prematurely tripping the comparator. *Linear Technology; $3.74 (1,000s).*

Reader Service No. 15

Software

Utilities boost DOS machines

PC-Kwik Power Pak's utilities speed the performance of 286-, 386-, and 486-based machines by employing a disk cache, screen and keyboard accelerators, and a print spooler. A data buffer in RAM boosts application speed. Accelerators speed cursor movement up to 126 cps and scrolling up to three times faster than standard. The print spooler stores data for the printer so the system can return to an application. A disk cache shares memory between PC-Kwik utilities and lends memory to application programs as needed. Other utilities in the set include a screen blanker that operates on a timer or with a hot key, and a command-line editor. *Multisoft; $70.*

Reader Service No. 16

Algebra system

Maple V for Amiga DOS is an interactive computer algebra system that delivers 3D postscript and image file format output graphics. Its mathematics library includes more than 2,000 functions, supported by an Arexx port. The program supports Commodore Amiga 1000, 2000, 2500, and 3000 on an Amiga DOS version 2.0 or higher with 2 Mbytes of RAM and 8 Mbytes free disk space. *Waterloo Maple Software; $450.*

Reader Service No. 17

Maple V sample output

Hspice graphic interface

Graphical Simulation Interface is a point-and-click, mouse-driven graphical user interface that provides interactive capabilities for quick analysis of Hspice simulations in an X-Windows environment. A machine-independent file format connects Hspice to GSI so users can run Hspice on a mainframe and view results graphically on a workstation.

GSI also provides concurrent simulation and wave review, point-and-click node property and selection, interactive curve measurement, automatic storage of last curve display, and flexible viewing with zoom, pan, multiple panels, and multiple simulation data. GSI supports most Unix X-Windows workstations. *Meta-Software; $2,000.*

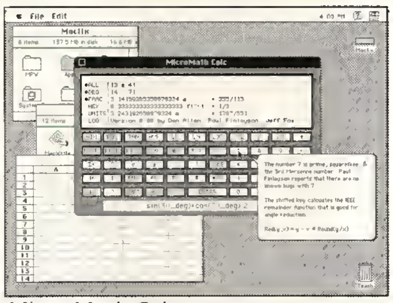
Reader Service No. 18

Scientific calculator for Mac

Micro Math Calc, a desk accessory calculator for the Macintosh, offers advanced features for programmers, mathematicians, and engineers. Real, complex, and Gaussian numbers can carry information on associated units and systems.

Users can enter and view numbers in binary, octal, decimal, hexadecimal, and with their corresponding ASCII characters. The calculator supports shifting operations, integer division, and logical bitwise operations such as Or, Not, And, and Xor. A bits function lets users quickly determine binary quantities. The calculator complies with the Standard Apple Numeric Environments and IEEE standards. Available for System 7, Micro Math Calc requires 100 Kbytes of memory. *Micro Math Scientific Software; $99.*

Reader Service No. 19



Micro Math Calc

Terminal emulators

KEAterm 420, version 2 emulates the DEC VT420 terminal for DOS machines running Windows. Emulated functions include multiple sessions and pages, double high/wide characters, and 132-column support. Extensions include user-definable keyboard mapping and attribute-mapped colors.

The software supports IBM's enhanced keyboard, DEC's LK250, and KEA's Power Station keyboard. Pull-down menus display in English, French, or German. Version 2 enhancements include network/file transfers and script language enhancements. Other features in this latest upgrade include interfaces for TCP/IP, KEAlink TCP, and Super Kermit file transfer protocols.

A second product, KEAterm 340, includes all the capabilities of KEAterm 420 and supports Regis, Tektronix, and sixel graphics. *KEA Systems; $245 (420), $395 (340).*

Reader Service No. 20

Signal processing hardware and software

20-MHz signal processor

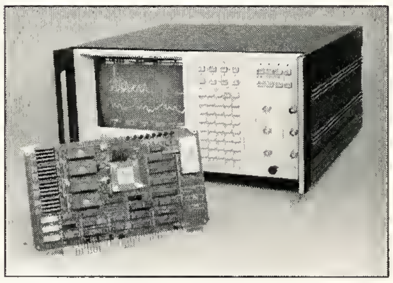
The 20-MHz ADSP-21020 floating-point DSP cycles in 50 ns and calculates a 1,024-point FFT in 0.96 ms. The manufacturer says its architecture, optimized for signal processing applications, suits it for image processing, graphics, radar and sonar, speech recognition, and advanced audio applications. The chip comes in commercial (0° to +85°C) and military (–55° to +125°C) temperature ranges, in a 223-lead pin grid array. *Analog Devices; $198 (1,000s).*

Reader Service No. 21

DSP speeds waveform analysis

Model 683 is a DSP add-on board that extends Analogic's Model 6100 Waveform Analyzer by a factor greater than 300. According to the company, the add-on board's 25-Mflops processor computes and displays an 8K-point FFT in milliseconds and a 16K × 16K cross-correlation analysis in one second. The speed allows real-time signal processing and spectral analysis. Model 683 uses a 32-bit floating-point DSP slaved to Model 6100's CPU. *Analogic; $2,995.*

Reader Service No. 22



Analogic's Model 683 and Model 6100

Smooths data

Data Smoother processes the scattered points of original data and creates a waveform while preserving any abrupt change in values. Release 2.0 for MS-DOS enables users to enter data from the keyboard or other ASCII sources. The program process 1,500 points of data in seconds and displays original and smoothed data in tables, alphanumeric strip chart, or graphically. Users can choose from predefined labels or create their own and save on disk. *Dynacomp; $50.*

Reader Service No. 23

16-bit input module

The SBX-416 is an isolated, 16-bit analog input module that uses a successive approximation analog-to-digital converter to process at up to 20 KHz. A software driver, compatible with Microsoft and Borland C compilers, generates a serial data stream. An optically isolated external trigger initiates data conversion. The module self-calibrates on start-up. *Systek; $559 (100s).*

Reader Service No. 24

Asynchronous servers

Asynserv2 and Asynserv8 (two- and eight-port units) allow LAN users to share modems; remote users can access the LAN and locally process under remote control. The two asynchronous communication servers support dial-in and dial-out communications at up to 57.6 Kbps per port, allowing 14.4-Kbps V.32bis modems with V.42bis data compression to run at full speed.

The systems include hardware and all necessary software to work with IPX or Net BIOS LANs. Other features include call-back security, host keyboard locking, screen blanking, multiple file transfer capabilities, script language, dialing directory, mail, chat mode, and pop-up menus. Asynserv measures 38.3 cm × 8.0 cm × 25 cm and feeds off a universal power supply (90 to 260V). *MNC International; $2,495 (Asynserv2), $3,895 (Asynserv8).*

Reader Service No. 25

Windows software

Windows development tool

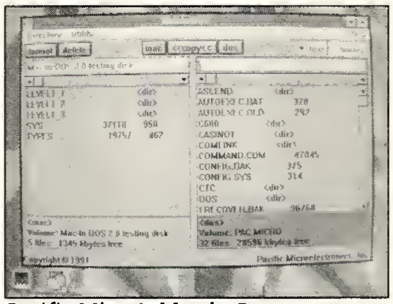
Desktop users in multiuser networks and client-server environments can develop applications in a Windows environment with Open Insight. It lets developers create database applications or link to SQL Server, Oracle, or other systems to create client-server applications. Open Insight includes development tools and an active data dictionary that gives users an integrated view of data sources, including dBase, ANSI SQL, SQL Server, ASCII, and DB2. Quarterly updates, add-on utilities, and a year of telephone technical support are included. *Revelation Technologies; $895.*

Reader Service No. 26

Mac-in-DOS for Windows

Mac-in-DOS, one of several programs that copy and convert files between Macintosh and DOS formats, is now available in a Windows format. Version 2.0 lets users run the program with Microsoft Windows 3.0. Function keys perform the main DOS file-handling functions, including changing subdirectories and deleting or copying files. *Pacific Micro; $99.*

Reader Service No. 27



```
LEVEL1_1          <dir>      ASLEND          <dir>
LEVEL1_2          <dir>      AUTOEXEC.BAT       328
LEVEL1_3          <dir>      AUTOEXEC.OLD       292
SYS        37778    958      C600          <dir>
TYPES      19757    467      LASINOT       <dir>
                             COMLINK       <dir>
                             COMMAND.COM      47845
                             CONFIG.BAK         375
                             CONFIG.SYS         314
                             CTC           <dir>
                             DOS           <dir>
                             FRECOVER.BAK     96768
<mac>                        <dos>
Volume: Mac-in DOS 2.0 testing disk   Volume: PAC MICRO
5 files  1345 kbytes free             32 files  28596 kbytes free
Copyright © 1991                      Pacific Microelectronics, Inc.
```

Pacific Micro's Mac-in-Dos

DOS or Windows applications

Developers can build Windows or DOS applications with APL Plus II/386, version 4. The program creates APL applications with graphical user interface for the Microsoft Windows 3.0 environment. Version 4 also interfaces to non-APL software, including most DOS software with an application programming interface for C programmers. It also includes an interface to Borland's Paradox Database Manager, a screen interface toolkit, and Super VGA (800 × 600) 256-color graphics support. *STSC; $1,700, $495 (upgrades).*

Reader Service No. 28

Mouse-driven help systems

Two windows programs let users create help systems or data validation entry screens for Microsoft Windows without writing code, by pointing and clicking with a mouse. Robo Help includes a tool palette that can simplify construction of help systems. It generates source codes for indexes, categories, defined terms, and hypertext files.

Magic Fields lets users develop data compilation fields by pointing and clicking to predefined data entry fields, and adding custom-designed fields. Users can specify fonts, colors, and a grayed 3D effect. It includes a library of numeric, text, alphanumeric, and monetary objects. *Blue Sky Software; $495 (Robo Help), $349 (Magic Fields).*

Reader Service No. 29

Document management

DOS users can retrieve or scan word processing, database, spreadsheet, or graphics files with DE/Cartes Document Manager. The icon-based system stores files with names up to 64 characters long. Each document can have an unlimited number of revisions online. Users can define their own hierarchy of storage.

One mouse click selects a file, a second accesses a user-defined note, a third loads the program. On remote systems, access is restricted to one user at a time per document, and unauthorized users can be locked out. DE/Cartes requires a DOS machine, MS-DOS 3.0, Microsoft Windows 3.0, graphics card, mouse, 5 Mbytes of hard disk space, and 2 Mbytes of RAM. *Desktop Engineering International; from $147.50 (introductory price).*

Reader Service No. 30

Display and scan peripherals

Video array with interface

Media Wall, an array of monitors with a computer interface, is a multimedia presentation system integrating computer animation, graphics, and text with full motion and still video. It includes an adapter card for the Macintosh, a satellite control unit containing special-effects hardware, and an array of stackable video monitors or projectors.

Users can display duplicate or different images in a variety of modes. Or they can display one high resolution (3,200 × 2,400) image tiled across the array. Monitors align up to 27 in a line or circle, or in a grid pattern up to five by five. *RGB Spectrum; $26,000 (interface board and controller box), $40,000 (complete unit with nine monitors). Macintosh not included.*

Reader Service No. 31

Pen base for desktops

Displaypad connects to desktop computers to make a pen-based system. Information written or drawn on the tablet simultaneously appears on the monitor. A cordless stylus senses tip pressure, height, and angle. The stylus' resolution is 1,270 lines per inch, and it has a data output of 200 points/s. The tablet features 640 × 480 resolution, 64 shades of gray, and a flush surface that allows smooth pen operation.

To install, a display card replaces the existing VGA card. Displaypad works with DOS, Macintosh, and Sun systems. *Cal Comp; $2,500.*

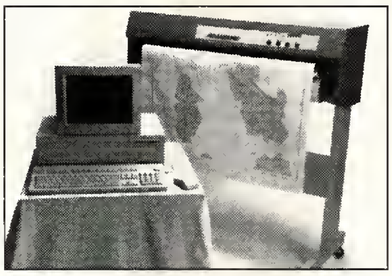
Reader Service No. 32

Cal Comp's Displaypad

Faster rasters

Two raster plotters for Macintosh applications achieve what the manufacturer says are the fastest speeds in their class. Model 2400, with a 24"-wide, D-size format, plots at 4" per second and can format and plot a D-size plot in under 40 seconds. Model 3600, with a 36"-wide, E-size format, outputs 2" per second. The 200-dpi, monochrome plotters use Microspot Mac Plot DMA driver software and a NuBus interface card to process output from Claris CAD, Mac Project, Pixel Paint, Super Paint, Power Point, Freehand, Dreams, and other Quickdraw programs. *Atlantek; $12,500 (2400), $14,500 (3600).*

Reader Service No. 33


Atlantek's Model 3600

Prints 30 pages per minute

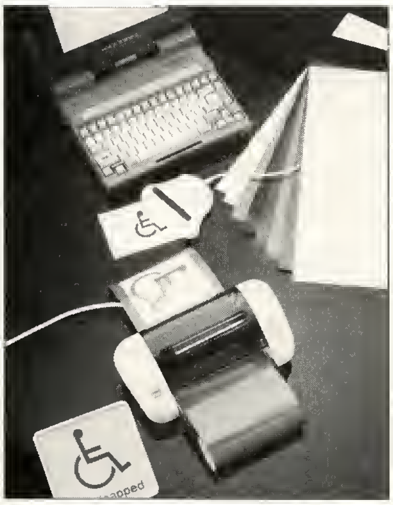
The LC-7030 nonimpact printer outputs 30 pages per minute. Two disk drives hold fonts; ambitious users can replace one with a 52-Mbyte hard disk for more storage. The controller supports HP PCL 5, Post Script, and DEC LN03 Plus emulations. Connects to a printer via RS-232 or RS-422 serial ports, Centronics parallel ports, Ethernet, TCP/IP, and twinaxial or coaxial cables. *Advanced Technologies International; $16,480 (simplex), $21,430 (duplex).*

Reader Service No. 34

In-house sign production

The Image Crafter creates multicolored graphic applications for signage and presentations. The package includes a desktop vinyl cutter/plotter, software, and a hand-held scanner. Users scan an image into their DOS machine (MS-DOS 3.1 or higher). There, they can manipulate and clean up the image in a window-based program, before sending the finished product to the plotter. *Kroy Sign Systems; $2,195.*

Reader Service No. 35


Kroy's Image Crafter

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 189 Medium 190 High 191

Product Summary

Joe Hootman
University of North Dakota

Manufacturer	Model	Comments	R.S.#
Boards			
Brooktrout Technology	TR112 fax card	PC/AT-compatible Twin Channel board contains two transceivers for multichannel facsimile applications. By taking advantage of direct-dialing services, an autorouting version sends incoming messages automatically to LAN fax-mail server users. $1,995 and $2,495 (autorouter). Quantity discounts available.	80
Rohm Corporation	Memory cards	Credit card-size and smaller SRAM, DRAM, mask ROM, one-time PROM, and flash-memory cards support applications in which small COB solid-state units can replace floppy disks. The 32-Kbyte to 6-Mbyte semicustom products promise 100-ns to 150-ns access times. From $35 to $950 (100s); 12 weeks ARO.	81
Star Tech	860 Edge add-in	With 32 to 128 Mbytes of DRAM and math/vector libraries, the Macintosh II i860 coprocessor accelerates the Pixar Mac Renderman photorealistic renderer, fitting into one Nubus slot. A developer's version supports the large floating-point operations required in scientific applications. $8,000 (32 Mbytes).	82
Traquair Data Systems	HEPC2 parallel processor	TMS320C40-based, PC/AT-compatible board supports parallel, image, and digital signal processing, as well as graphics computations. Supporting up to three TIM-40 TMS daughter boards, HEPC2 provides up to 200 Mflops of floating-point perfomance (1.1 billion operations/s). From $1,539 (mother board); from $1,500 (TIM-40s).	83
Software			
Micro Touch Systems	Power Keypad	PC Unmouse software lets Microsoft Windows and DOS users control the cursor and execute macros by touching a keypad marked on a template inserted beneath the Unmouse glass surface. To click, users press down on the glass. Free upgrade to Unmouse owners.	84
Motorola	Smart Model for 68302	Behavioral-level simulation model for the 68302 integrated multiprotocol processor lets designers develop, debug, and optimize the hardware operation of 302-based designs before commiting to physical prototypes. Smart Model features intelligent error-checking, user-defined timing, and VHDL interoperability. $4,000 for one-time technical licensing fee, plus Logic Automation library license fee.	85
Systems			
Anorad	Anoguide AG-12, AG-14	Controller- and linear motor-equipped series boosts speeds to 100 ips and up to 120 lbs. of force at a 25-percent duty cycle. For noncontact operation in an industrial environment, the Anoline brushless, iron core coil assembly attaches to a moving	86

Manufacturer	Model	Comments	R.S.#
		slide while the stationary permanent magnets are mounted to the stage's base plate.	
Ariel	IRCAM workstation	Designed for compute-intensive applications on the Next Cube, this signal-processing workstation uses two i860 RISCs combined with a DSP56001 to perform at 160 Mflops and 93.5 MIPS. The CPOS/FTS real-time operating system provides protected multi-tasking kernel, memory management, file I/O, interprocessor communications, and process management facilities. $14,995; university discounts available.	87
Neotronics/Laser Monitoring Systems	TE-TC temperature controller	Controller with built-in safety features supports semiconductor devices that need to be operated at lower than ambient temperatures. A four-digit, seven-segment LED shows either set or measured values of temperature, resistance, and current.	88
Nighthawk Electronics	DXS-16 data exchanger	Several computers can share a number of peripherals (printers, modems, file servers) with the 16-Mbyte DXS-16 as a LAN alternative. Each unit maintains 500,000-bps speeds on up to 16 serial, parallel, or 3270 coaxial/twin-axial ports.	89
PI Systems	Infolio pen computer	Pen-based, 2.9-lb., 9 × 10 × 1.2-in., PCMCIA-memory computing tool features a 640 × 480-pixel VGA-quality, reflective LCD. Infolio integrates a Cal Comp cordless stylus and Motorola 68331-based hardware with PDX-framework database software, a graphical user interface, and task-specific application software for mobile information collection and management solutions. $1,895.	90
Miscellany			
I-Con Industries/ Antel Corporation	Multiwire backplane	Futurebus+ backplane in A, B, and F profiles with 64- and 128-bit data widths comes in 5-, 9-, and 14-slot configurations. The multilayer board uses precision discrete wires rather than etched circuits for signal interconnection and eliminates signal layers and the number of corresponding etched voltage and ground planes required for impedance reference.	91
Multiaccess Computing	MCC-1000F adapter	Frame relay service adapter card provides Macintosh II users with multiple, presubscribed virtual connections across metropolitan or wide-area networks. The one-board controller occupies one Nubus I/O slot on the system board and runs at T1 or fractional T1 rates over a T1 facility. $2,995; 60 days ARO.	92
Parsytec	TIP I/O bus boards	TIP series expands I/O on transputer-based parallel processing systems, transmitting data simultaneously, in parallel, to multiple transputer nodes via its broadcast function. The broadcast rate equals $n \times 100$ Mbytes/s, where n equals the number of receivers. Series includes the TIP-VPU/T8 T805 processor board, TIP-MFG monochrome frame grabber, and the TIP-CGD color graphic display board. From $4,600 each; 3 or 4 weeks ARO.	93

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

District Manager: D. Rodney Brooks; Tel: (415) 905-0260; Fax: (415) 896-1512.

Recruitment and Classified Advertising: D. Rodney Brooks; Tel: (415) 905-0260; Fax (415) 896-1512.

Director of Sales: Randall L. Stickrod, 544 Second St., Suite 200, San Francisco, CA 94107; Tel: (415) 905-0260; Fax: (415) 896-1512.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

	RS #	Page #
Advanced Technologies Int'l	34	93
Analog Devices	21	91
Analogic	22	91
Anorad	86	94
Antel Corp.	91	95
Ariel	87	95
Atlantek	33	93
Blue Sky Software	29	92
Brooktrout Technology	80	94
Cal Comp	32	93
Desktop Engineering Int'l	30	92
Dynacomp	23	92
GIGATEC SA	1	C.IV
I-Con Industries	91	95
KEA Systems	20	91
Kroy Sign Systems	35	93
Linear Technology	15	90
Meta-Software	18	91
Micro Math Scientific Software	19	91
Micro Touch Systems	84	94
Mitsubishi	13	89
MNC International	25	92
Motorola	14, 85	90, 94
Multiaccess Computing	92	95
Multisoft	16	91
Neotronics/Laser Monitoring Systems	88	95
Nighthawk Electronics	89	95
Pacific Micro	27	92
Paradigm	11	89
Parsytec	93	95
PI Systems	90	95
Revelation Technologies	26	92
RGB Spectrum	31	93
Rohm Corp.	81	94
STSC	28	92
Star Tech	82	94
Systek	24	92
Texas Instruments	10, 12	89
Traquair Data Systems	83	94
Waterloo Maple Software	17	91

THE FOLLOWING INFORMATION IS AVAILABLE:

Contact the Publications Office;
to facilitate handling, please request by number.

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Compmail electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204
  (requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

IEEE COMPUTER SOCIETY®
A member society of the
Institute of Electrical and Electronics Engineers, Inc.

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes seven magazines and five research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

COMPUTER SOCIETY OFFICES

Headquarters Office
1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone: (202) 371-0101
Fax: (202) 728-9614

Publications Office
10662 Los Vaqueros Cir.
PO Box 3014
Los Alamitos, CA 90720-1264
Membership and General Information:
(714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

European Office
13, Ave. de L'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Fax: 32 (2) 770-85-05

Asia/Pacific Office
Ooshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone: 81 (3) 3408-3118
Fax: 81 (3) 3408-3553

CommNexus

Presents

The LiSBUS™ Async I/O System

A solution of the 1990's for today's data transmission problems.

Outstandingly Simple and Reliable because **LiSBUS™** is based on a breakthrough technology which uses the impedance of the bus cable to replace binary addresses. Consequently, data transmission management is greatly simplified and much more reliable than today's equivalent systems which require expensive software, hardware, and personnel investments.

Outstandingly Practical because it is easy to install and operate. No special tools, workbench, or electronics expertise are needed. Anyone can be up and running in minutes. Just plug in the external modules and configure the system with the user-friendly **LiSBUS™ Link Control Software**. Each external module measures only around 2in. by 2in.

Outstandingly Flexible because a user can connect up to 60 peripherals or computers to a controlling computer through their RS-232C (COM) ports. To add peripherals, just extend the bus cable and add modules.

At an Unbeatable Price because at $650* for the **LiSBUS™ Starter Pack**, no alternative offers all these advantages combined into one product without spending a much higher amount. The **Starter Pack** includes all the user needs to connect four peripherals and a complete set of **LiSBUS™ Software Development Tools** to create custom applications.

LiSBUS™ Async I/O System: A product of our **CommNexus™** line of communication systems. GIGATEC is committed to offering products and servicing its customers in the best tradition of Swiss quality. We provide our customers with:

- **Technical Support.** Registered buyers can obtain technical support from our qualified engineers.
- **Users and Developers Group.** Organized for encouraging software developments using the products of the **CommNexus™** family.

For more information and ordering contact:

In the USA and Canada:

Toll Free (800) 945-3002
(excl. Hawaii)
Mon.-Fri. 9am - 9pm EST

GIGATEC (USA). Inc.
871 Islington Street
P.O. Box 4705
Portsmouth, NH 03802-4705 USA
Tel. (603) 433-2227
Fax (603) 433-5552

In Europe:

GIGATEC SA
Ch. des Plans-Praz
1337 Vallorbe SWITZERLAND
Tel. 41 21 843 37 36
Fax 41 21 843 33 25

* specifications and prices subject to change without prior notification

Visa and MasterCard/EuroCard accepted. - CommNexus™ and LiSBUS™ are trademarks of GIGATEC SA.

Reader Service Number 1